Exhibit 1

IN THE UNITED STATES DISTRICT COURT
FOR THE SOUTHERN DISTRICT OF NEW YORK

Hermès International and
Hermès of Paris, Inc.,

Plaintiffs,

vs.

Mason Rothschild,

Defendant.

Civil Action No. 22-cv-00384 (JSR)

Expert Report of Kevin D. Mentzer, Ph.D.

August 4, 2022

Expert Report of Kevin D. Mentzer, Ph.D.

Contents

Introduction

I am Kevin D. Mentzer. I am the Endowed Chair of Data Science and Emerging technologies at Nichols College. I have extensive experience in blockchain technologies. My work address is 121 Center Rd, Dudley, MA 01571.

Professional Qualifications

I have an M.S. in Information Technology and a Ph.D. in Business Administration with a focus in Business Analytics from Bentley University. I specialize in emerging technologies including blockchain technologies. From 2015 to 2022 I served as an Assistant, then Associate Professor with tenure, at Bryant University where I designed and taught classes in blockchain technology, information systems, analytics, and data science and served as co-founder and faculty advisor to the Bryant blockchain club. I secured funding for, and oversaw the deployment of, the club's crypto mining lab. In January 2022 I joined Nichols College as the Endowed Professor of Data Science and Emerging technologies where I launched the Intelligent Automation concentration, redesigned the analytics MBA curriculum, and designed the Data Science undergraduate curriculum. I have 30+ years relevant experience in software development and database design.

I regularly publish peer-reviewed academic research in the fields of blockchain technology and social media analytics and have been invited to give talks and serve on panels related to blockchain in both academic and industry settings. I passed the Certified Blockchain Solutions Architect exam by the Blockchain Training Alliance in 2018.

Peer-Reviewed Publications

Mentzer, K., Price, J., Powers, E., Lavrenchuk, N. (Forthcoming - 2022) Examining the Hype behind the Blockchain NFT Market. *Issues in Information Systems* 23(x)

Mentzer, K., Frydenberg, M., & Yates, D. J. (2020). Teaching Applications and Implications of Blockchain via Project-Based Learning: A Case Study. *Information Systems Education Journal*, *18*(6), 57-85.

Phan, L. Li, S., Mentzer, K. (2019) Blockchain Technology and the Current Discussion on Fraud. *Issues in Information Systems* 20(4)

Peer-Reviewed Conference Papers

Mentzer, K., Frydenberg, M., Yates, D. (2019) Teaching Applications and Implications of Blockchain via Project-Based Learning: A Case Study. *EDSIGCON 2019*

Presentations and Panels (academic and industry)

Mentzer, K. (2021) "Introduction to Blockchain for Internal Auditors" Presented to Mohegan Sun, Uncasville CT

Mentzer, K., Paul, M. (2021) "Blockchain 101 – Understanding the Current NFT Market" Presented at Analytics without Borders, Virtual.

Mentzer, K., Yates, D. (2018) "Teaching Blockchain: Technology, Applications, and Implications" Panel Participant at EDSIGCON, Norfolk, VA.

Mentzer, K., Gough, M. (2018) "The Impact of Cryptokitties on the Ethereum Blockchain" Presented at Northeast Decision Sciences Institute annual meeting, Providence, RI.

Expert Witness Experience

This is the first case I have been involved with as an expert witness.

Background and Assumptions

I understand that Hermès International and Hermès of Paris, Inc. have filed civil action against Mason Rothschild for appropriating Hermès' trademarks in the launch of the MetaBirkins NFTs.

It is my understanding that Mason Rothschild conceived of, and was responsible for, the launch of the MetaBirkins NFTs on the Ethereum blockchain ("MetaBirkins NFT Project"). I therefore assume that Mason Rothschild earned all revenue from the MetaBirkins NFT Project identifiable from the blockchain. This revenue includes all minting revenue and all royalties from sales made on the OpenSea, LooksRare, and Rarible Marketplaces.

My understanding is that Mason Rothschild conceived of, and was responsible for, the launch of the Baby Birkin NFT prior to the launch of the MetaBirkins NFT Project.

My understanding is that Mason Rothschild conceived of, and was responsible for, the launch of the I Like You You're Weird (ILYYW) NFT collection in March 2022, after the launch of the MetaBirkins NFT Project.

My understanding is the masonrothschild.eth is the digital wallet owned by Mason Rothschild.

All analysis is based on transactions through 7/1/2022.

Assignment

I was tasked to answer the following questions:

1. What is a MetaBirkins NFT?
2. How were MetaBirkins NFTs Generated?
3. How much did Rothschild earn from the MetaBirkins NFTs?
4. Have MetaBirkins NFT holders received any other benefits?
5. Did the MetaBirkins NFTs differ from the BabyBirkin NFT?
6. Did the MetaBirkins NFTs differ from the ILYYW NFTs?
7. Can more "MetaBirkins" NFTs be minted?
8. Can the "MetaBirkins" name be changed?
9. Can MetaBirkins NFTs be destroyed (i.e. burned)? Altered? Clawed back?
10. Are there any other remedies?
11. How can the MetaBirkins NFTs be used?
12. Could Rothschild make MetaBirkins NFTs Wearable?

To answer these questions, I identified the smart contracts involved with the MetaBirkins NFT Project, identified the digital wallets involved with the project, and analyzed all transactions related with the project. To accomplish this, I searched for the MetaBirkins collection on LooksRare, found a recent sale, used the link from that sale to the transaction as seen on Etherscan.IO, and used that transaction to identify the primary smart contract associated with the MetaBirkins NFTs. After identifying the smart contract, I used that address to load the smart contract into the Remix development environment located at remix.ethereum.org. I used the commonly available public tools of etherscan.io and Remix development environment to both analyze the smart contract code and to study the details behind each transaction.

I looked at the transactions related to the BabyBirkin NFT sold in May 2021. The BabyBirkin NFT was sold through the basic.space marketplace. I was able to locate those transactions through the use of basic.space to get the transaction ID and information gained from the basic.space Twitter account which mentioned that it was on the FLOW Blockchain. I used the site flowscan.org to see the transactions related to the BabyBirkin.

Finally, I looked at the transactions and smart contract related to the I Like You You're Weird NFT collection that launched in March 2022. Using a recent sale as found on the OpenSea Marketplace, I located the smart contract. Using the same tools (etherscan.io and Remix) I analyzed the smart contract and transaction history.

Terminology

Below are definitions of terminology use in this report.

Airdrop – An airdrop is a distribution of cryptocurrency or tokens (fungible or non-fungible tokens) that are sent to a digital wallet for free. This is frequently done as a promotion or added value as a result of owning a token or participating in some event.

Blockchain – A blockchain is a decentralized digital ledger spread across many computers linked through a peer-to-peer network. All transactions are validated and saved on the digital ledger that is publicly available[1] with the entire transaction history available and non-changeable (i.e. immutable). The two largest public blockchains are Bitcoin and Ethereum. Public blockchains have a native currency that is used to pay for transactions and maintain the network. The currency for the Bitcoin blockchain is called Bitcoin while the currency for the Ethereum blockchain is called Ether.

Burn – To burn an NFT means that an individual NFT is transferred to a non-existent wallet making the NFT no longer usable by anyone.

Claw Back – Claw back refers to functionality that could be incorporated into a smart contract enabling the smart contract to pull a token from another account. It is a way for the smart contract owner to force the transference of an NFT without the NFT owner's permission.

Closing Price – The price of Ether is constantly fluctuating, in order to understand historical values it is common to refer to the value that the Ether had at the end of the day. This report uses a frequently used closing price as posted on etherscan.io (etherscan.io, n.d.-a).

Crypto Wallets – A crypto wallet is a digital wallet used to store cryptocurrency and tokens. Crypto Wallets allow for the secure transfer of cryptocurrency and tokens and are used to prove to the blockchain that you are the owner of those assets. Wallet IDs used on Ethereum have a 40 hexadecimal length (e.g. 0xa061982d4b087d911db8399a641187df945d48d0).

[1] While there are private blockchains, only public blockchains apply here since all projects mentioned in this document reside on a public blockchain.

EIP – EIP is an abbreviation for a Ethereum Improvement Proposal. Due to the decentralized nature of Ethereum, any changes to Ethereum can be made by anyone through an EIP. EIP's have different types including ERC which is used for application-level standards.

ERC-721 – Currently the most popular NFT standard is ERC-721[2] (which was the foundation on which MetaBirkins NFTs were implemented). All NFTs on the Ethereum blockchain that adopt the ERC-721 standard must have a core set of functions that dictate the common functionality used by all NFTs. For example, a NFT contract has to have a transferFrom function that dictates how one NFT can be moved from one wallet to another wallet. This ensures that all NFTs have a common set of functionality. The contract itself is then loaded to the blockchain which ensures that the code cannot be changed (unless a change is allowed for in the contract). The saving of the contract to the blockchain also generates the collection which records the token name and symbol. Because of the immutable nature of the blockchain (i.e. unalterable history) smart contracts cannot be changed.[3]

Ether/ETH – Ether, commonly abbreviated ETH, is the official native cryptocurrency for the Ethereum blockchain. Ether is used to pay the fees related to Ethereum and is the most popular currency used for buying and selling NFTs.

Ethereum – Ethereum is a decentralized blockchain platform run by a peer-to-peer network that executes and verifies smart contracts as well as maintains a distributed ledger. Ethereum is the second largest (based on the market cap of the native cryptocurrency) public blockchain behind Bitcoin. However, unlike Bitcoin, Ethereum allows for the generation of Non-Fungible Tokens, making it the blockchain of choice for many NFT projects including the MetaBirkins NFT. Ethereum is the overarching term related to all functionality of the Ethereum architecture including how the peer-to-peer network is managed, how the Ether cryptocurrency is minted, how changes are made to the architecture, and how the Ethereum blockchain is recorded.

Ethereum Blockchain – the Ethereum Blockchain represents one piece of Ethereum, namely the ledger that records all the transactions and smart contracts.

Gas – Gas is the fees that are paid to post a transaction to the blockchain. These fees are distributed to those who help support the blockchain.

LooksRare Marketplace – Decentralized marketplace launched in Jan 2022 and a relatively new site for NFT trading. This marketplace offers a rewards program but has been criticized that majority of sales are wash sales (selling the NFT to themselves) to gain rewards.

Metaverse – The Metaverse is still evolving but generally thought of as a combination of virtual reality and augmented reality worlds accessed through a browser or headset. It is envisioned as a virtual space where people could work and play.

Minting – Minting is the generation of an individual NFT from a collection. This is when the unique NFT is generated and transferred to its first owner. This could be done entirely by the creator of the collection, or it can be delegated to others in some fashion. One common approach is to create a list of

[2] ERC-1155 is the other popular NFT standard.
[3] They can potentially be replaced from a point forward if the smart contract has been designed to be replaceable.

wallets that are given permission to generate one or more NFTs from the collection. This process of creating an approved list of creators is called Whitelisting.

To mint NFTs that utilize a whitelist, the whitelist members most likely go to the NFT's website, connects their digital wallet to the website ensuring they are the owner of that wallet, and uses funds in their digital wallet to mint a new NFT. The website then interacts with the smart contract which generates the NFT token, transfers ownership of the transaction to that wallet, and posts the transaction to the blockchain.

NFT – NFT is an abbreviation for Non-Fungible Token. Non-Fungible means that it is unique and cannot be replaced with something else. Fungible means the token is not unique. 1 Bitcoin is exactly like every other bitcoin so it does not matter which one you have, they are fungible. Your Superman #1 comic book is unique (condition, ownership history, etc.) and I simply substitute another comic book for your Superman #1, that means it is non-fungible. For digital tokens this means that on NFT is unique from all others. NFTs could me a song, a movie, a piece of digital art, or even a token representing ownership of a real-world item.

NFT Collection – An NFT collection, as used in this document, represents the architecture of the NFT. The collection has a smart contract that defines how each individual NFT within the collection behaves. The NFT Collection also has an ownership which allows that individual the rights to set up royalties or make changes to the contract (where allowed). Those changes then take effect for each individual NFT within the collection. To generate an NFT, first the NFT Collection must be set up. This is done through programming code, usually written in the programming language Solidity. The program tells Ethereum that there is a new token being generated and defines the business rules associated with that token. These business rules specify the functionality that is available for that token related to the generation, buying and selling, and destroying of those tokens. Once the NFT collection is generated on the blockchain, the minting, or generation, of the individual NFTs can occur

NFT Marketplace – A NFT Marketplace is a third-party site that offers a trading platform for NFTs. Depending on the marketplace they may offer sales, auctions, and a place to make offers on NFTs that are not for sale. Marketplaces also offer collection owners the chance to set a royalty for sales of the NFTs in that collection that occur through that marketplace.

Off-chain – Any information that is not stored directly in the blockchain transactions. Off-chain data represents data that could potentially change and has a lower level of trust.

On-chain – Any information that is accessible from the blockchain transactions. Historical data stored on-chain cannot be changed meaning it is trusted more than off-chain data.

OpenSea Marketplace – Currently the largest NFT Marketplace. The majority of MetaBirkins NFTs sold through the OpenSea marketplace which recently surpassed $20 Billion in trading volume.

.png – A .png file is a common image format favored by NFTs due to the high color quality, the fact that is it free to use, and the ability to have a transparent background making removal of the background easy.

Rarible Marketplace – A community-owned marketplace for Ethereum NFTs approaching $300 Million in trading volume.

Royalties – Royalties are payments back to the collection owner (assumed to be the original creator) on any subsequent sale of the NFT. Royalties can be either defined in the smart contract or set on the individual NFT marketplaces. When an NFT collection has a royalty, then when the individual NFT is sold then a percentage of that sale is sent to the collection owner.

Smart Contract – Code that is placed on the blockchain that dictates some set of business rules. In the case of an NFT, the smart contract(s) usually dictate how NFTs are generated, how they can be transferred or sold, and whether the NFT can be destroyed (burned). Smart contract can be found using their 40-digit hexadecimal code (e.g. 0x566b73997F96c1076f7cF9e2C4576Bd08b1A3750 is the address for the MetaBirkins NFT contract).

Solidity – The coding language used for the MetaBirkins smart contracts.

$-Values – All $ values in this report are based on the closing price of Ether for the date of that transaction as posted on etherscan.io.

UTC – UTC is an abbreviation for Coordinated Universal Time and is the standard time used by the Ethereum blockchain in recording times as stored on the blockchain.

Whitelist – An NFT “whitelist” is a list of crypto wallets that are pre-approved for minting an NFT. A whitelist can either be on-chain (meaning the list of wallets are recorded in a transaction on the blockchain) or off-chain (meaning the list is managed somewhere else and some technique to prove you are on the whitelist occurs in the smart contract). When a wallet is whitelisted, it may not guarantee that they can still mint an NFT (for example, the collection could sell out before that wallet has a chance to mint), but the collection creator can control this process to ensure everyone whitelisted will get an NFT (for example, if only 100 NFTs are going to be generated then limit the whitelist to 100 wallets that can each generate only 1 NFT).

Wrapped Ether/WETH – Wrapped ether is a token on the Ethereum blockchain that is pegged 1:1 with Ether. Because WETH is pegged to ETH, for the purpose of this report, WETH is treated the same as ETH.

1. What is a MetaBirkins NFT?

A MetaBirkins NFT is a is a token on a Ethereum blockchain that represents ownership of a unique item (a MetaBirkins NFT). The MetaBirkins NFT collection utilizes the ERC-721[4] standard. The MetaBirkins NFT collection is managed by a smart contract (see Appendix D) which has a set of standard functions that manage the entire collection.

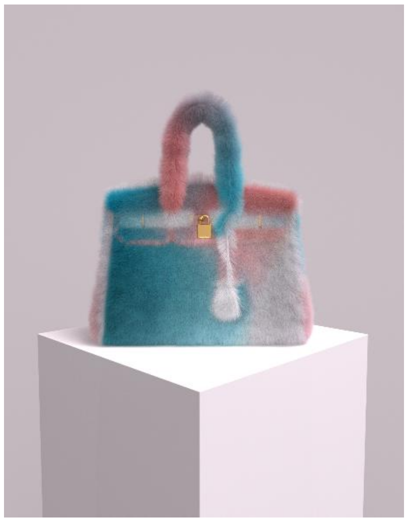


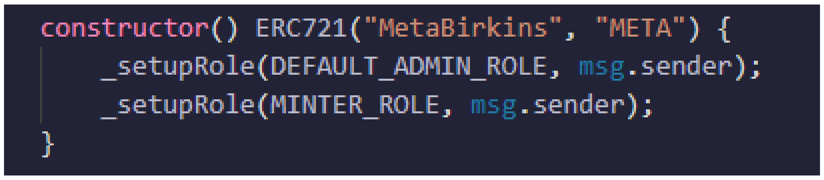
Figure 1. Sample MetaBirkins NFT

The image in Figure 1 is a visual representation of one of the MetaBirkins NFTs. Since ownership of a MetaBirkins NFT is tracked on the blockchain, no one can change the record of ownership without interacting with the smart contract. The MetaBirkins NFT smart contract (because it is based on ERC-721) can only manage one NFT, in this case the MetaBirkins NFT. While smart contracts that implement the ERC-721 standard must have certain functions, the people who code the smart contract can make it unique and work according to their own needs for various functionality.

2. How were MetaBirkins NFTs Generated?

The generation of the MetaBirkins NFTs, from the point of the blockchain, begins with the creation of the smart contract which occurred on Dec 2nd, 2021 at 6 AM UTC. To identify the smart contract involved with the generation of the MetaBirkins collection, I began by finding the most recent sale from the LooksRare Marketplace (which allows for the viewing of historical transaction). I then used the link from that transaction on LooksRare to navigate to the Etherscan.io site (an industry standard tool to search for transactions on the Ethereum blockchain) to retrieve the transaction for this sale. This sale was for MetaBirkins #31 which occurred on Mar 11th, 2022 (Ethereum transaction #0x7e9ecd5b6770654cf951dba41a681ede868cf6d1d6e4a2da58512c2a1e615786). Using this transaction, I was able to identify the smart contract that was used to transfer this token from the seller to the buyer. This led me to the primary MetaBirkins contract (0x566b73997F96c1076f7cF9e2C4576Bd08b1A3750).

Using the smart contract address, along with the software development tool Remix, I was able to look at the Solidity code for this contract. Remix is a commonly used online integrated development environment used to write, compile, and debug smart contracts for Ethereum. In particular, I was looking for the code for the constructor function (see Figure 2).



```
constructor() ERC721("MetaBirkins", "META") {
    _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
    _setupRole(MINTER_ROLE, msg.sender);
}
```

Figure 2. MetaBirkins Constructor Code

[4] While ERC721 is not the only NFT standard (see also ERC1155), this report focuses on that standard since it was used as the standard to generate the MetaBirkins NFT.

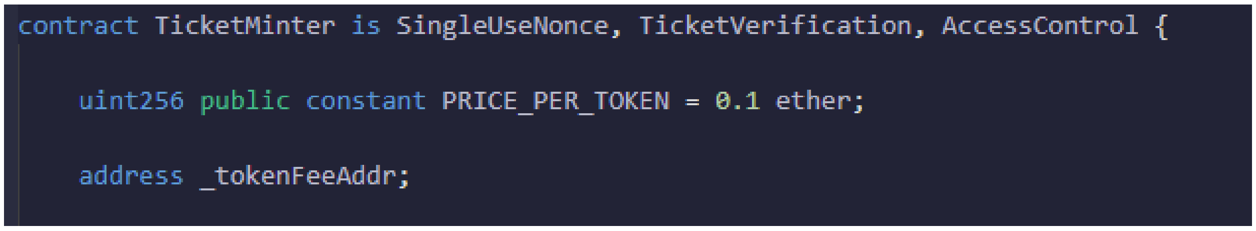
This code tells me several important things. First, that the MetaBirkins NFTs were set up using the ERC721 token type. This is the most common token type for NFTs and indicates that this contract is used solely for the management of the MetaBirkins NFTs (no other NFTs can be managed through this contract). Second, the NFTs were assigned the name "MetaBirkins" and the Token symbol "META". Finally, I can see that the admin role and minter role default to the wallet responsible for creating this contract.

By looking at the first transaction linked to this smart contract (0x58c3b471397a2a9399222dd6d73b76d679cd4f3267a17824ad7e9714ea93e087) I am able to see that the creator was wallet: 0xcd9d4dd3e066a77039607a82588365b75a1f4778. This transaction occurred on Dec 2nd, 2021 at 6:00:26 AM UTC[5] and represents the time the smart contract was established on the blockchain and the initial generation of the MetaBirkins collection.

I then searched for all transactions that used this contract and found 242 transactions. In analyzing the 242 transactions that occurred under this contract, I was able to identify the minting smart contract that was used to generate all the individual MetaBirkins NFTs. The minting contract (0xa18667035c5C925889b1548E94D40854C83370d2) was created on Dec 2nd, 2021 at 6:08:30 AM UTC by the same wallet that created the master contract.

In reviewing the code in the minting contract, I could determine that the minting price of each MetaBirkins NFT was set to .1 ETH (see Figure 3) and that there was code in place for both a whitelist minting and a public minting (only the whitelist minting ended up being used).

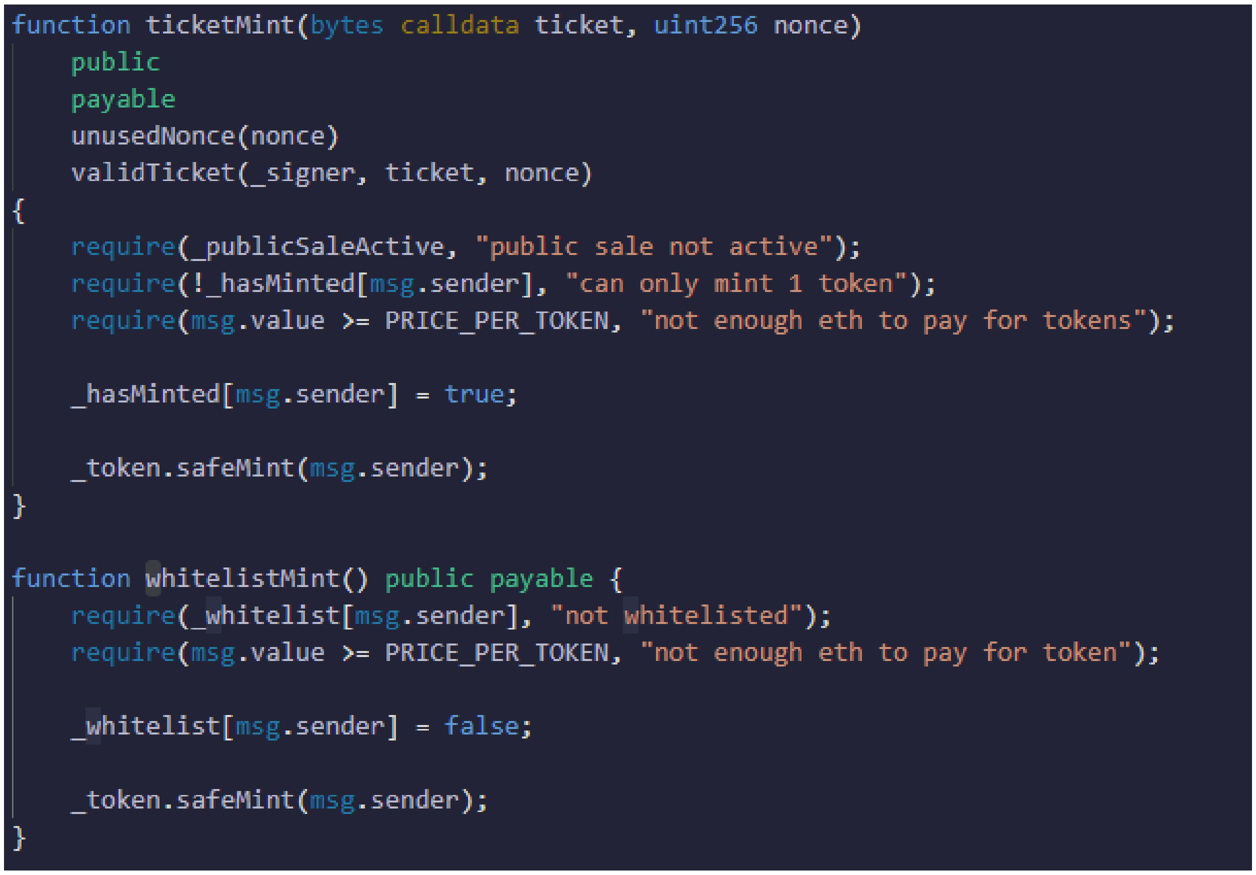
[5] UTC or Coordinated Universal Time uses 0̂ longitude and is not adjusted for daylight savings time.



```
contract TicketMinter is SingleUseNonce, TicketVerification, AccessControl {

    uint256 public constant PRICE_PER_TOKEN = 0.1 ether;

    address _tokenFeeAddr;
```



```
function ticketMint(bytes calldata ticket, uint256 nonce)
    public
    payable
    unusedNonce(nonce)
    validTicket(_signer, ticket, nonce)
{
    require(_publicSaleActive, "public sale not active");
    require(!_hasMinted[msg.sender], "can only mint 1 token");
    require(msg.value >= PRICE_PER_TOKEN, "not enough eth to pay for tokens");

    _hasMinted[msg.sender] = true;

    _token.safeMint(msg.sender);
}

function whitelistMint() public payable {
    require(_whitelist[msg.sender], "not whitelisted");
    require(msg.value >= PRICE_PER_TOKEN, "not enough eth to pay for token");

    _whitelist[msg.sender] = false;

    _token.safeMint(msg.sender);
}
```

Figure 3. Portions of the Minting Smart Contract

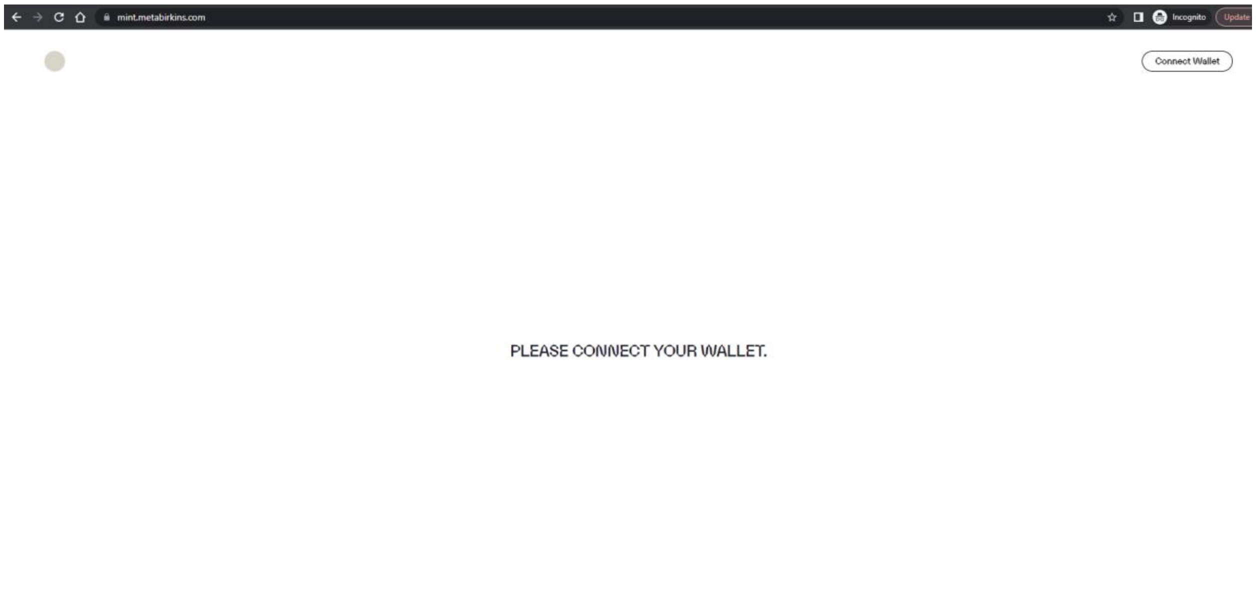
As recorded in transaction 0x2377c8711a3f6893f689717dee8971926e6605ca8d4fd910b4fd3e02b52fa3f1 at 6:08:46 AM UTC on Dec 2nd, 2021, the minting contract was assigned the Minter Role from the master contract. This was executed by the same wallet that created both contracts. This tied the minting contract back to the master contract and allowed the minting process to begin.

How were the MetaBirkins NFTs Distributed?

The distribution of the MetaBirkins NFTs, from the blockchain perspective, consisted of generating a list of valid wallets that would be allowed to generate an NFT and then the actual generation of the individual NFTs. Since the MetaBirkins NFTs used an on-chain whitelist, I was able to find all the transactions related to adding wallets to the whitelist.

The creation of the whitelist began on Dec 2nd, 2021 at 6:08 AM UTC (as recorded in transaction 0xb34f05db72c48cb4d4a45ca79015ae9c38f9224689934955b0b2845d2ef93aae) with the adding of the first wallet (0x278db415b3c969e789e1ec9e80e1e001a5dcee82) to the whitelist. At 10:58 PM UTC an additional 98 wallets were added to the whitelist. Finally, at 11:00 PM UTC the last wallet[6] (before minting begins) was added to the whitelist. This was wallet (0xc57112fb1872130a85ecf29877dd96042572a027).

Now that the list of valid wallets was set, the generation of the individual NFTs began. In the case of MetaBirkins, the minting of the 100 individual NFTs were initiated by those on the whitelist. Based on documents produced by Mr. Rothschild and provided to me, this minting appears to have occurred through the website mint.metabirkins.com. The whitelisted user would go to that page and was instructed to connect their digital wallet to the site as shown in Figure 4.



PLEASE CONNECT YOUR WALLET.

Figure 4. Initial screen for mint.metabirkins.com as of July 30, 2022

Once the wallet was connected, they were brought to the minting page as shown in Figure 5. You can see in the upper right-hand corner that wallet "0x27…e82" (abbreviated) is connected to the site. The user would see the message "Press the button below to mint your Birkin now" and would select the "Mint Token" button. In the background, the page would then call the "whitelistMint" function in the minting smart contract as shown in Figure 3 which would result in the MetaBirkins NFT being generated and assigned to this initiating wallet.

[6] There were more wallets added to the whitelist on Dec 3rd, 2021.

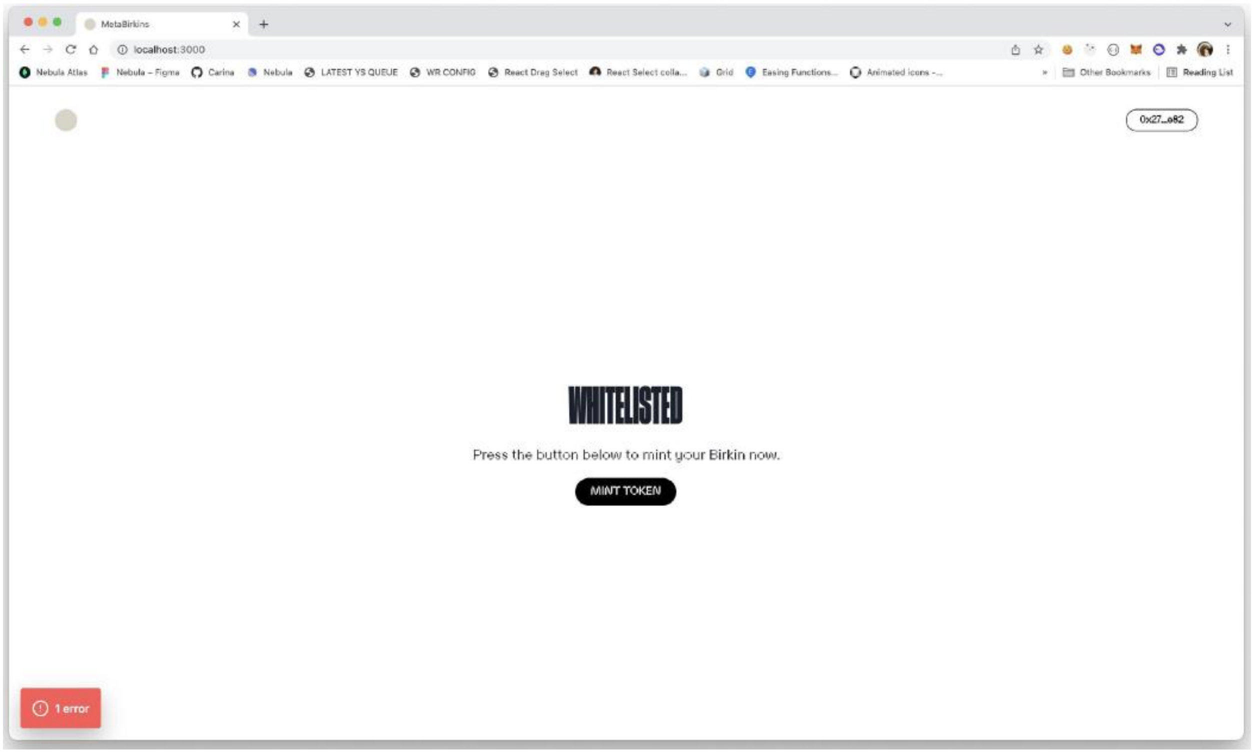
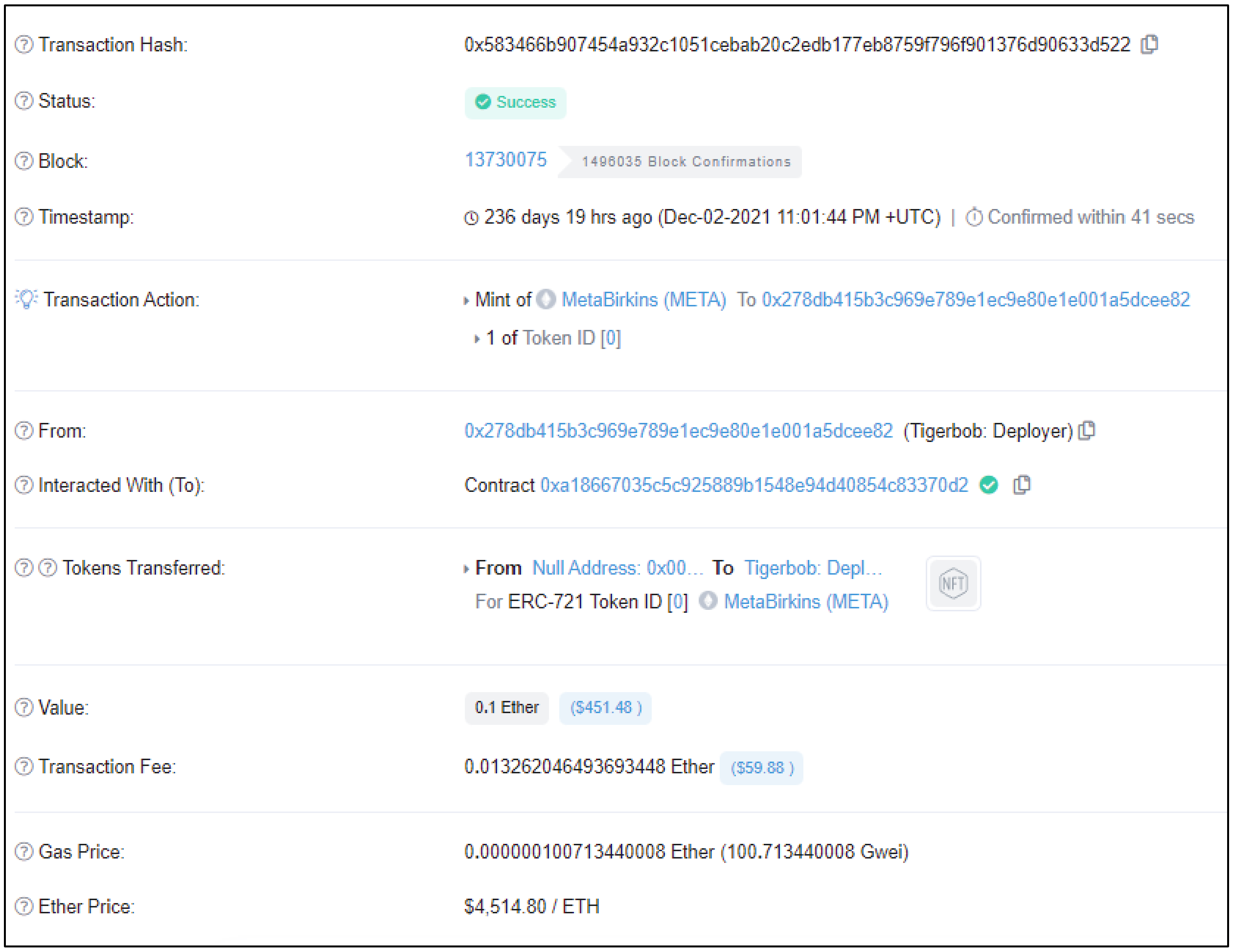


Figure 5. Website mint.metabirkins.com after wallet connected (Rothschild007790)

For example, when MetaBirkins #0 was generated (see Figure 6) we can see that wallet 0x278db415b3c969e789e1ec9e80e1e001a5dcee82 called the minting contract passing in .1 ETH ($451.48) plus paying the gas fee of .01326 ETH ($59.88) and in return was given the NFT MetaBirkins #0. The gas fee goes to those responsible for maintaining the Ethereum blockchain and the .1 ETH payment is deposited into the contract (to be extracted later). As can be seen in Figure 6, the price of Ether that day was 1ETH = $4,514.80. So, this wallet paid $451.48 plus the gas fee of $59.88 charged to validate and record the transaction.



Transaction Hash:	0x583466b907454a932c1051cebab20c2edb177eb8759f796f901376d90633d522
Status:	Success
Block:	13730075 1496035 Block Confirmations
Timestamp:	236 days 19 hrs ago (Dec-02-2021 11:01:44 PM +UTC) | Confirmed within 41 secs
Transaction Action:	Mint of MetaBirkins (META) To 0x278db415b3c969e789e1ec9e80e1e001a5dcee82 1 of Token ID [0]
From:	0x278db415b3c969e789e1ec9e80e1e001a5dcee82 (Tigerbob: Deployer)
Interacted With (To):	Contract 0xa18667035c5c925889b1548e94d40854c83370d2
Tokens Transferred:	From Null Address: 0x00... To Tigerbob: Depl... For ERC-721 Token ID [0] MetaBirkins (META)
Value:	0.1 Ether ($451.48)
Transaction Fee:	0.013262046493693448 Ether ($59.88)
Gas Price:	0.000000100713440008 Ether (100.713440008 Gwei)
Ether Price:	$4,514.80 / ETH

Figure 6. Sample Transaction as seen on Etherscan.io

The minting of the NFTs occurred in sequential order starting with #0 and ending with #99. Overall, minting began at 11:01 PM UTC on Dec 2nd, 2021 and ended at 7:01 PM UTC on Dec 3rd, 2021 when NFT #99 was generated. At this time no individual images were linked to the NFTs, so the people minting those NFTs would not see specific MetaBirkins, instead they were buying an unknown image of a MetaBirkins NFT which would be revealed later.

The minting pattern of the 100 MetaBirkins NFTs is shown in Figure 7. The x-axis shows the time that each MetaBirkins NFT was minted and the y-axis shows the MetaBirkins NFT #. The graph starts with #0 on the far left and ending with #99 on the far right. We see that the lower numbers were minted prior to the bulk of the minting with a gap between those low numbers and a large surge of minting starting around 2 AM UTC and ending around 3 AM UTC. Then there is a large gap in time before the final ones were minted shortly before the minting contract was destroyed.

I discovered some evidence that favoritism may have occurred in the minting process. In my experience, based on my participation in NFT transactions, lower number NFTs often hold more value that higher numbers. #1 is worth more than #10 which is worth more than #100. The minting pattern caught my attention because the low numbered MetaBirkins NFTs were generated first and then there were gaps before the bulk of the minting occurred. The minting of MetaBirkins #0 and #1 went to wallets that were also treated a bit differently when added to the whitelist (one was added first as a separate transaction and the other was added last as a separate transaction). Later, I discovered the #1 was transferred to masonrothschild.eth.

The minting contract was destroyed at 7:03 PM UTC, 2 minutes after MetaBirkins #99 was generated. At the time of contract destruction, the contract held 10 ETH ($42,193.20). The minting revenue was distributed to masonrothschild.eth (9 ETH / $37,973.88) and 0xcd9d4dd3e066a77039607a82588365b75a1f4778[7] (1 ETH / $4,219.32) wallets on Dec 3rd, 2021 at 7:03 PM UTC when the contract was destroyed.

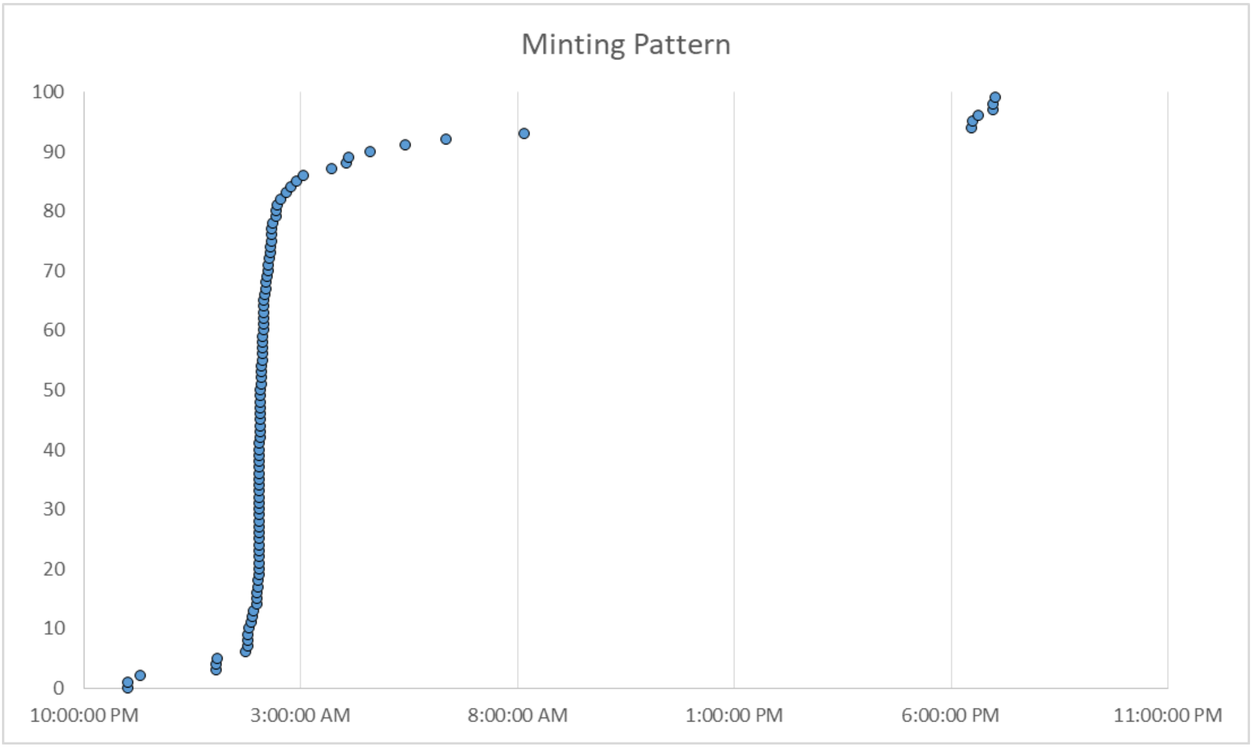


Figure 7. MetaBirkins Minting Timeline

Sales and Transfers

Once an individual NFT is minted, it is available to be transferred or sold. In fact, we see sales begin even before the minting was complete or the images of the NFTs were unveiled. The image of each bag is not stored on the blockchain, but a link to the image is on the blockchain. This is a common approach for NFTs since gas fees are paid based on the amount of data stored on the blockchain, so if the entire image was stored on the blockchain then the gas fees related to that NFT would be much higher. However, by storing the images off-chain, they can be changed over time.

The MetaBirkins images are linked through a decentralized repository called IPFS (InterPlanetary File System). This is a peer-to-peer network for sharing data without a centralized repository. There is a master directory that is set using the baseURI function defined in the primary smart contract. Each individual image is then located in a subdirectory based on the number of the NFT.

[7] At 8:14 PM UTC on the same day, this wallet transferred 1 ETH to masonrothschild.eth.

At 12:16 AM UTC on Dec 3rd, 2021, for a brief period of time the image on the OpenSea Marketplace associated with each MetaBirkins NFT was set to the video shown in Figure 8 (the ending image is also shown). Anyone looking at the MetaBirkins NFTs on the OpenSea Marketplace between 12:16 AM UTC on Dec 3rd, 2021 and 1:57 AM UTC on Dec 3rd, 2021 would have seen this video for all the minted MetaBirkins NFTs.

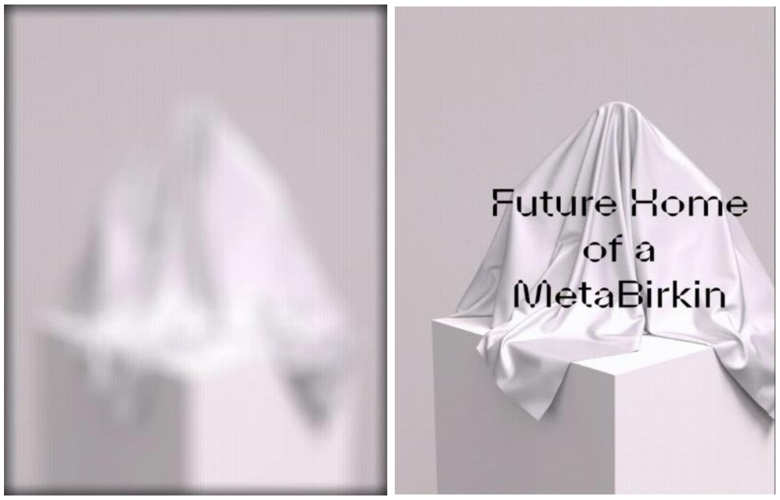


Figure 8. First Veiled Object Image

At 1:57 AM UTC on Dec 3rd, 2021, prior to anyone enabling sales on OpenSea Marketplace, the image appearing on the OpenSea Marketplace for all the MetaBirkins NFTs changed from the movie file shown in Figure 7 to the still image shown in Figure 8. From 1:57 AM UTC on Dec 3rd, 2021 to 7:34 PM UTC Dec 3rd, 2021 anyone shopping for a MetaBirkins NFT on the OpenSea Marketplace would see the "veiled object" image along with the MetaBirkins NFT #. In other words, the only thing making one bag different than another in the collection was the NFT # (0 through 99).

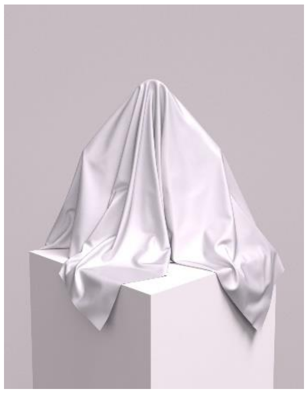
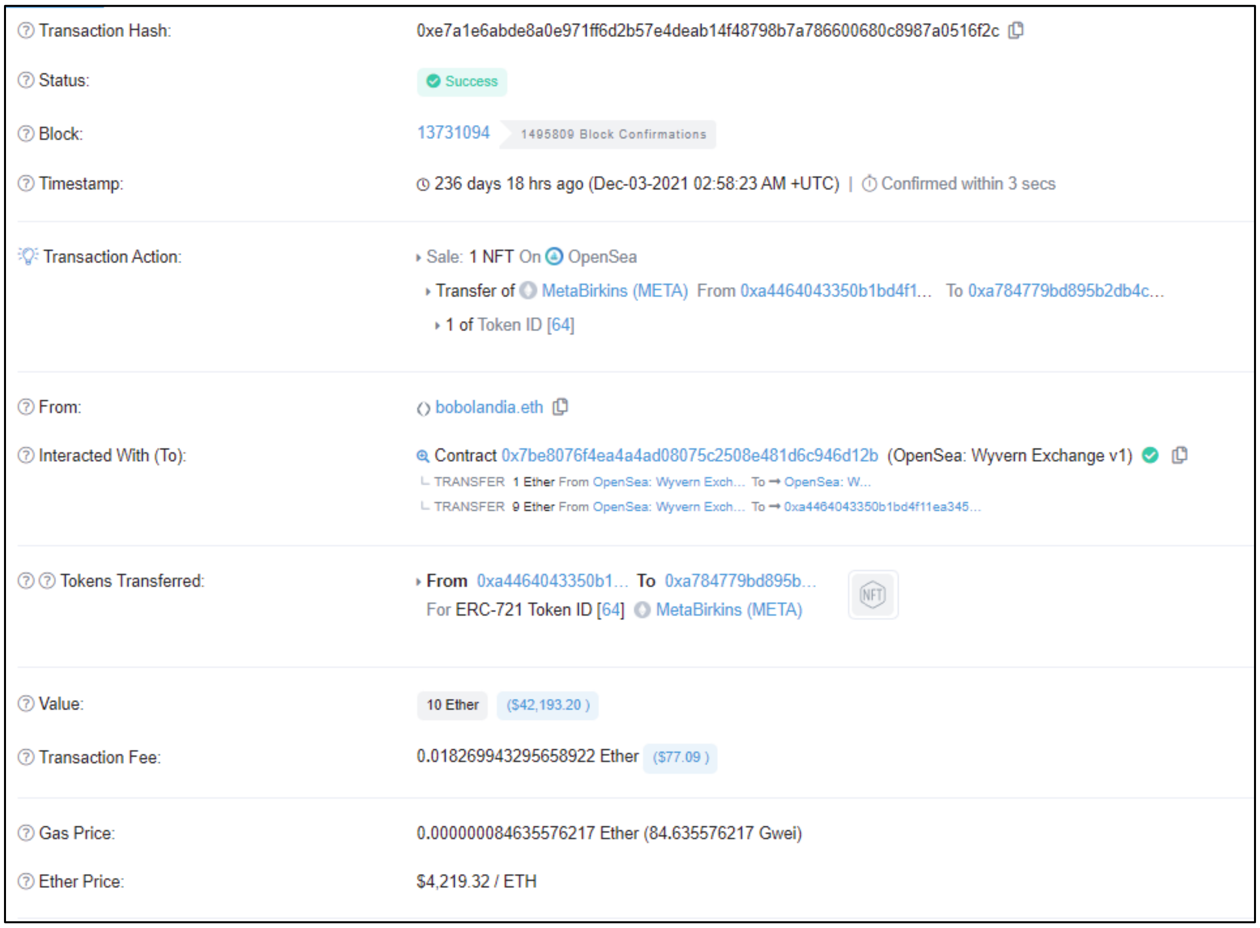


Figure 9. Veiled Object Image

For any MetaBirkins NFT holder to sell on a third-party marketplace like OpenSea.io , the owner of the NFT has to give permission to OpenSea Marketplace to broker on their behalf. What this does is confirm that the seller agrees to the terms and services of the site and allows the marketplace to wrap their own smart contract around the sale, which allows them to charge fees and royalties not covered in the underlying NFTs' smart contract. We see these permissions being granted starting at 2:07 AM UTC on Dec 3rd, 2021 while the minting was still occurring (59 MetaBirkins NFTs had been minted by this time).

The first sale of a MetaBirkins NFT occurred at 2:58 AM UTC on Dec 3rd, 2021, when #64 sold for 10 ETH ($42,193.20). At this time, the veiled object image was still in effect and 86 MetaBirkins had been minted (i.e. minting was still ongoing). The transaction details are shown in Figure 10.



Transaction Hash:	0xe7a1e6abde8a0e971ff6d2b57e4deab14f48798b7a786600680c8987a0516f2c
Status:	Success
Block:	13731094 1495809 Block Confirmations
Timestamp:	236 days 18 hrs ago (Dec-03-2021 02:58:23 AM +UTC) | Confirmed within 3 secs
Transaction Action:	Sale: 1 NFT On OpenSea Transfer of MetaBirkins (META) From 0xa4464043350b1bd4f1... To 0xa784779bd895b2db4c... 1 of Token ID [64]
From:	bobolandia.eth
Interacted With (To):	Contract 0x7be8076f4ea4a4ad08075c2508e481d6c946d12b (OpenSea: Wyvern Exchange v1) TRANSFER 1 Ether From OpenSea: Wyvern Exch... To → OpenSea: W... TRANSFER 9 Ether From OpenSea: Wyvern Exch... To → 0xa4464043350b1bd4f11ea345...
Tokens Transferred:	From 0xa4464043350b1... To 0xa784779bd895b... For ERC-721 Token ID [64] MetaBirkins (META)
Value:	10 Ether ($42,193.20)
Transaction Fee:	0.018269943295658922 Ether ($77.09)
Gas Price:	0.000000084635576217 Ether (84.635576217 Gwei)
Ether Price:	$4,219.32 / ETH

Figure 10. MetaBirkins NFT First Sale Transaction

As can be seen in Figure 10, the wallet named bobolandia.eth purchased MetaBirkins #64 for 10 ETH ($42,193.20) and paid an extra .01827 ETH ($77.09) in gas fees. Of that 10 ETH, 9 ETH was transferred to the wallet starting 0xa4464… and 1 ETH was transferred to the OpenSea Marketplace wallet. OpenSea Marketplace charges a 2.5% fee on all NFT sales. This means that any amount more than 2.5% has been collected on behalf of the collection owner as a royalty. In this case we can see that 10% was collected meaning 2.5% went to OpenSea Marketplace and 7.5% was collected as a royalty. Royalties[8] can only be set up by the collection owner with the royalties collected being paid out on a monthly basis to the account specified when royalties are set up. Since this transaction has a royalty included, we know that the owner of the contract set this up prior to any sales occurring on OpenSea Marketplace.

[8] See: https://docs.opensea.io/docs/10-setting-fees-on-secondary-sales (*10. Setting Fees on Secondary Sales*, n.d.)

Sales of "veiled image" MetaBirkins NFTs continued until the "unveiling" event that occurred at 7:34 PM UTC on Dec 3rd, 2021. The unveiling represents when each NFT received their unique image. For example, at this time the image as seen on OpenSea Marketplace for bag #99 would have changed from the veiled image shown in Figure 9 to the unique image shown in Figure 11.

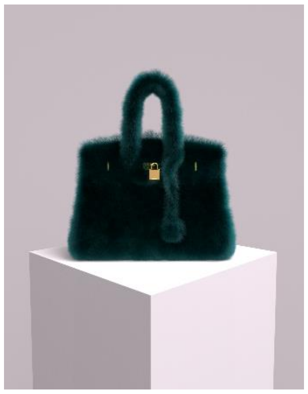


Figure 11. One of the MetaBirkins after the Unveiling

After the unveiling event, each Metabirkins NFT received a name and a link to the image file.

```
{
  •  name: "99",
  •  image: "ipfs://QmeRGrrgxP1u9muNZsxsfTvXq6HDQ442mD3GbdB9Ka1s5P/0-20.png"
}
```

In the case of bag #99 (as shown in Figure 11), the name given to it was "99" and the image file had the name 0-20.png. I found no obvious relationship between the image number and the related bag number. Table 1 shows some the last 4 bags in the collection and their linked image file name. So bag #99 was linked to image #0-22.

Bag #	Image File Name
99	0-20.png
98	0-155.png
97	0-102.png
96	0-1.png

Table 1. MetaBirkins bag # and image file name.

There are two items of interest here. First, the 0-xx numbering scheme suggests that there may be a series # associated with the file scheme and these MetaBirkins NFTs were considered Series 0. Second, that more than 100 images appear to have been generated since we see numbers larger than 100 (ex. Bag #98 is linked to image file #0-155).
Overall, there were 16 sales of MetaBirkins that occurred before the unveiling event took place. Those 16 sales ranged from 4.5 – 10 ETH with an average of 6.82 ETH ($28,786). There were 8 sales on OpenSea Marketplace in the 48 hours after the unveiling with an average of 4.2375 ETH ($17,617) per sale. Overall, there were 48 sales on OpenSea Marketplace between Dec 3rd, 2021 and Jan 15th, 2022. These sales totaled 263.1889 ETH ($1,088,461.93).

Sales also occurred on the LooksRare and Rarible Marketplaces. Unlike OpenSea Marketplace, the transaction setting up the royalty agreement is stored on-chain for these two platforms.

We can see, through the Ethereum transaction (0x5e2cd9d102a6c1e8eb2e0c2402d613b0e19a5bda59e2311965d1a0de28ea077a), that masonrothschild.eth set up a 7.5% royalty for the MetaBirkins NFT collection on the Rarible Marketplace on Dec 20th, 2021 at 7:40:31 PM UTC.

We can see, through the Ethereum Transaction (0x37c8075ff3919473e6499840a54209f1f2fe0f7e90651cb221ecf0c40665370f) that masonrothschild.eth set up a 7.5% royalty for the MetaBirkins NFT collection on the LooksRare Marketplace on Jan 10th, 2022.

There were 5 sales between Dec 26th, 2021 and Jan 16th, 2022 on the Rarible Marketplace ranging from 1.5 – 3.8 ETH for an average price of 3.02 ETH ($10,152.14).

There were 4 sales on the LooksRare Marketplace between Jan 23rd, 2022 and Mar 11th, 2022 ranging from 1.5-3.5 ETH for an average price of 3 ETH ($7,982.40).

In addition to sales, there were also transfers of NFTs occurring. These transfers may represent a gift, a moving of an NFT between 2 wallets controlled by the same person, or the result of some off-chain event. In any case, no royalties would have been collected on any transfer.

3. How much did Rothschild earn from the MetaBirkins NFTs?

To measure how much Rothschild received from the MetaBirkins NFT Project according to records on the blockchain, I considered the following:

1. Receipt of ETH as a result of the minting process
2. Receipt of ETH from royalties as a result of subsequent sales
3. Receipt of MetaBirkins NFTs directly

What I did not consider were any transaction that may have occurred involving off-chain events such as:

1. Exchanging assets to be placed on the MetaBirkins NFT whitelist
2. Exchanging assets to get a favorable MetaBirkins NFT minting spot
3. Reciprocity for minting benefits with another NFT launch
4. Exchange of assets to receive a specific or rare image

Next, I will cover each of the items I did consider.

Revenue from the Minting Process

The minting process was the actual generation of the individual MetaBirkins NFT. As discussed earlier, this process started on Dec 2nd, 2021 and ended at 7:01 PM UTC on Dec 3rd, 2021. Those who were given the opportunity to mint an NFT paid .1 Ether for that privilege. In exchange they received the NFT. This process generated all 100 MetaBirkins NFTs with a total of 10 ETH being paid to the Minting Contract. The contract would hold this money until it was withdrawn. On Dec 3rd, 2021 at 7:03 PM UTC, 2 minutes after the last MetaBirkins NFT was minted, 9 ETH (worth $40,629.24 based on the closing price of ETH on Dec 3rd, 2021) were transferred from the minting contract to the masonrothschild.eth wallet and 1

ETH (worth $4,514.36) was transferred to the 0xcd9d4dd3e066a77039607a82588365b75a1f4778 wallet. The total value earned from the minting process was 10 ETH worth $45,143.60.

Royalties

A second way that Mason Rothschild earned money was through royalties from all subsequent sales that occurred on a marketplace that had royalties set up. Royalties are a way for the original creator of the NFT to receive a percentage of sales that occur after the initial sale (minting). There are two common methods for establishing a royalty. The first method is to build a royalty transaction directly into the smart contract. The second method is to set up a royalty on the marketplace where sales may occur. I examined the smart contract code to look for any reference to a royalty and determined that smart contract was not designed to handle royalties so I knew that royalties would have to be set up somewhere outside the smart contract.

To identify royalties that had been earned on various marketplaces I examined each transaction that occurred when any of the MetaBirkins NFTs changed hands. When an NFT changes hands, it is recorded on the blockchain along with the selling price if the NFT was sold. To identify all sales transactions, I started at the Token Tracker page on Etherscan.io for the MetaBirkins NFTs (https://etherscan.io/token/0x566b73997f96c1076f7cf9e2c4576bd08b1a3750). This shows me a list of all transfers of the MetaBirkins NFTs. From this list I separated out the sales transactions from transfers that were made without exchanging cryptocurrency (transfers would have no royalty since no money was paid). I then looked up each transaction to record the selling price transferred to the seller along with any other funds transferred.

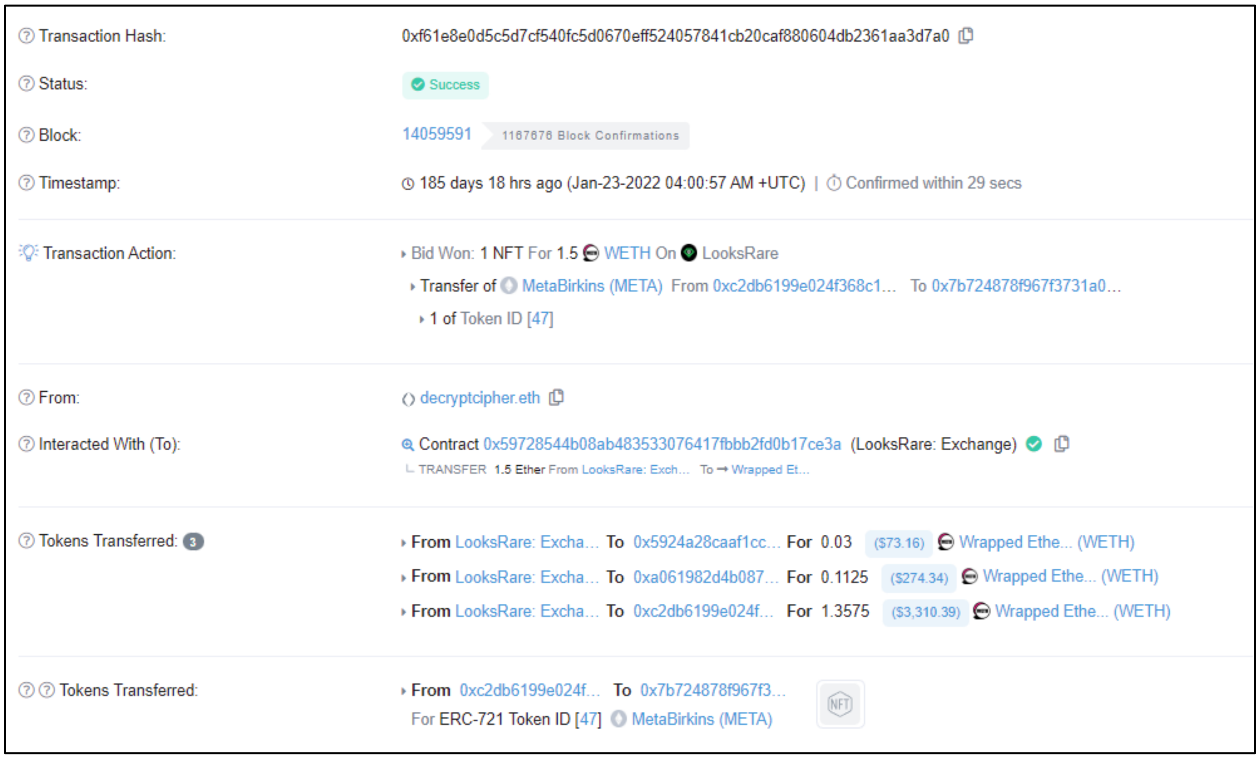
When the NFT is established on the NFT marketplace (i.e. OpenSea, Rarible, and LooksRare), the owner of the contract can set up a royalty to be paid on any transactions occurring on that platform. This is done on the website for each marketplace by the collection owner. Different marketplaces allow for different percentages to be collected, but in the case of the MetaBirkins NFTs, 7.5% was the royalty across all three marketplaces that sold any MetaBirkins NFT. While the setting for OpenSea Marketplace is off chain (meaning we cannot see the transaction that set up the royalty) it can be inferred from sales. The royalty setting for Rarible and LooksRare are recorded on the blockchain (and can be confirmed through each transaction).

Overall, there were a total of 57 sales (for complete list see Appendix B) that occurred between Dec 3rd, 2021 and Jul 1st, 2022. In analyzing each transaction, I determined that there was a 7.5% royalty charged on 44 of those transactions and 0% royalty charged on the remaining 13 transactions.[9] The total sales value totaled $1,171,152.22 with royalty revenue totaling $69,029.30. Sales occurred on three marketplaces: Opensea, Rarible, and LooksRare and I will next discuss how I confirmed the royalty for each marketplace and sale.

Overall, there were 4 sales on the LooksRare Marketplace between the dates of Jan 23rd, 2022 and Mar 11th, 2022. Sales on the LooksRare Marketplace itemizes the royalty as part of the transaction. LooksRare Marketplace charges a 2% fee for sales (see:

[9] The contract owner, Mason Rothschild, changed the royalty on OpenSea from 7.5% to 0% on Dec 4th, 2021 and changed back to 7.5% on Dec 9th, 2021.

https://docs.looksrare.org/about/rewards/platform-fee). Figure 12 shows the details behind one of the LooksRare Marketplace transactions as seen on Etherscan.io:



Transaction Hash:	0xf61e8e0d5c5d7cf540fc5d0670eff524057841cb20caf880604db2361aa3d7a0
Status:	Success
Block:	14059591 1187676 Block Confirmations
Timestamp:	185 days 18 hrs ago (Jan-23-2022 04:00:57 AM +UTC) | Confirmed within 29 secs
Transaction Action:	Bid Won: 1 NFT For 1.5 WETH On LooksRare Transfer of MetaBirkins (META) From 0xc2db6199e024f368c1... To 0x7b724878f967f3731a0... 1 of Token ID [47]
From:	decryptcipher.eth
Interacted With (To):	Contract 0x59728544b08ab483533076417fbbb2fd0b17ce3a (LooksRare: Exchange) TRANSFER 1.5 Ether From LooksRare: Exch... To → Wrapped Et...
Tokens Transferred: 3	From LooksRare: Excha... To 0x5924a28caaf1cc... For 0.03 ($73.16) Wrapped Ethe... (WETH) From LooksRare: Excha... To 0xa061982d4b087... For 0.1125 ($274.34) Wrapped Ethe... (WETH) From LooksRare: Excha... To 0xc2db6199e024f... For 1.3575 ($3,310.39) Wrapped Ethe... (WETH)
Tokens Transferred:	From 0xc2db6199e024f... To 0x7b724878f967f3... For ERC-721 Token ID [47] MetaBirkins (META)

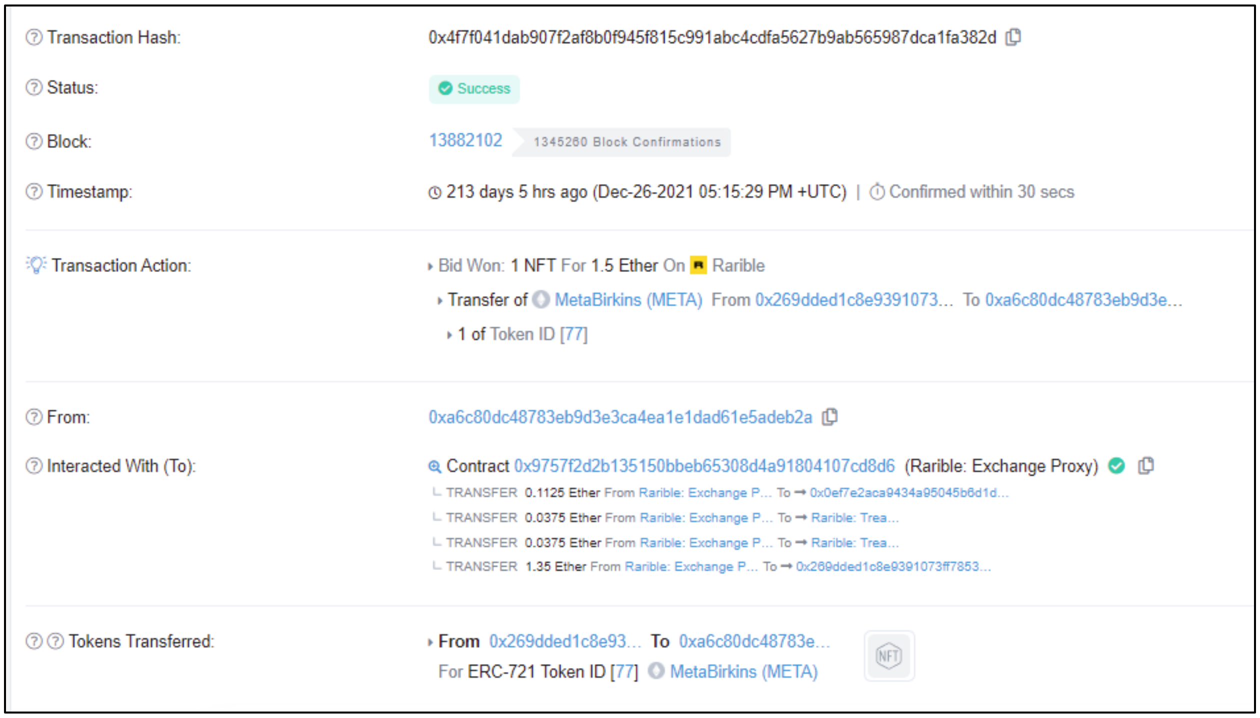
Figure 12. Sample Sale on LooksRare Martketplace

As can be seen in Figure 12, decryptcipher.eth purchased MetaBirkins #47 on the LooksRare Marketplace for 1.5 ETH. That 1.5 ETH is distributed as follows:

- 1.3575 ETH were transferred to the seller
- .03 was the LooksRare fee of 2%
- 0.1125 (7.5%) was transferred directly to the masonrothschild.eth wallet (0xa061982d4b087d911db8399a641187df945d48d0).

I repeated this process for each sale on the LooksRare Marketplace confirming that each sale on the LooksRare Marketplace included the 7.5% royalty paid directly to the masonrothschild.eth wallet.

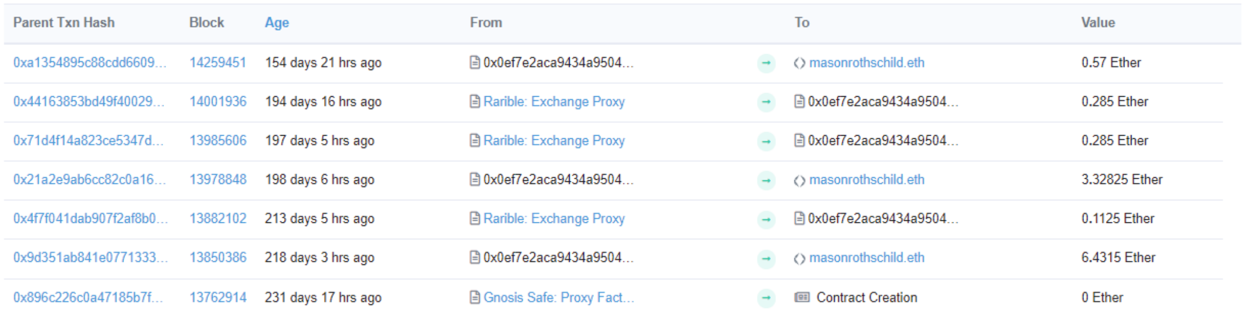
Next, I looked at the Rarible Marketplace transactions. There were 5 sales on the Rarible Marketplace between the dates Dec 26th, 2021 to Jan 16th, 2022. Sales on the Rarible Marketplace also itemize the royalty as part of the transaction. Prior to Jun 20th, 2022 (all transaction occurred prior to this date), the Rarible Marketplace charged a 2.5% seller fee and a 2.5% buyer fee (see: https://rarible.com/blog/lower-fees/). Figure 13 is an example of one of the Rarible Marketplace transactions.



Transaction Hash:	0x4f7f041dab907f2af8b0f945f815c991abc4cdfa5627b9ab565987dca1fa382d
Status:	Success
Block:	13882102 1345260 Block Confirmations
Timestamp:	213 days 5 hrs ago (Dec-26-2021 05:15:29 PM +UTC) | Confirmed within 30 secs
Transaction Action:	Bid Won: 1 NFT For 1.5 Ether On Rarible Transfer of MetaBirkins (META) From 0x269dded1c8e9391073... To 0xa6c80dc48783eb9d3e... 1 of Token ID [77]
From:	0xa6c80dc48783eb9d3e3ca4ea1e1dad61e5adeb2a
Interacted With (To):	Contract 0x9757f2d2b135150bbeb65308d4a91804107cd8d6 (Rarible: Exchange Proxy) └ TRANSFER 0.1125 Ether From Rarible: Exchange P... To → 0x0ef7e2aca9434a95045b6d1d... └ TRANSFER 0.0375 Ether From Rarible: Exchange P... To → Rarible: Trea... └ TRANSFER 0.0375 Ether From Rarible: Exchange P... To → Rarible: Trea... └ TRANSFER 1.35 Ether From Rarible: Exchange P... To → 0x269dded1c8e9391073ff7853...
Tokens Transferred:	From 0x269dded1c8e93... To 0xa6c80dc48783e... For ERC-721 Token ID [77] MetaBirkins (META)

Figure 13. Sample Sale on Rarible Marketplace

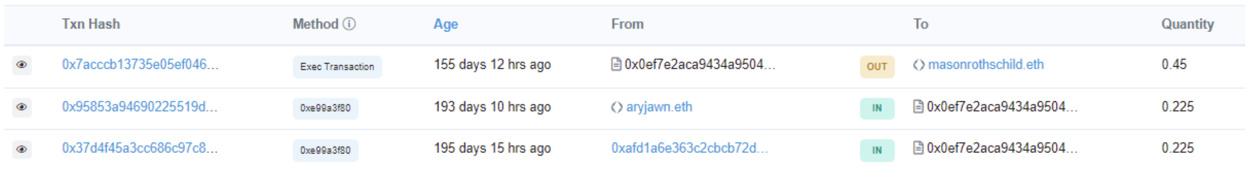
In this transaction we can see that the wallet 0xa6c80dc48783eb9d3e3ca4ea1e1dad61e5adeb2a purchased MetaBirkins #77 for 1.5 ETH[10] on Dec 26th, 2021. We can see both the buyer fee and the seller fee of 2.5% (.0375 ETH) transferred directly to a Rarible Treasury wallet. The transfer of .1125 ETH to the wallet 0x0Ef7E2aCa9434a95045b6d1D92Bc774F711C0ca8 represents the 7.5% royalty. In looking at the transactions from this wallet, as seen in Figures 14 and 15[11] below, this wallet then transferred these funds to masonrothschild.eth shortly after each sale:



Parent Txn Hash	Block	Age	From		To	Value
0xa1354895c88cdd6609...	14259451	154 days 21 hrs ago	0x0ef7e2aca9434a9504...	→	masonrothschild.eth	0.57 Ether
0x44163853bd49f40029...	14001936	194 days 16 hrs ago	Rarible: Exchange Proxy	→	0x0ef7e2aca9434a9504...	0.285 Ether
0x71d4f14a823ce5347d...	13985606	197 days 5 hrs ago	Rarible: Exchange Proxy	→	0x0ef7e2aca9434a9504...	0.285 Ether
0x21a2e9ab6cc82c0a16...	13978848	198 days 6 hrs ago	0x0ef7e2aca9434a9504...	→	masonrothschild.eth	3.32825 Ether
0x4f7f041dab907f2af8b0...	13882102	213 days 5 hrs ago	Rarible: Exchange Proxy	→	0x0ef7e2aca9434a9504...	0.1125 Ether
0x9d351ab841e0771333...	13850386	218 days 3 hrs ago	0x0ef7e2aca9434a9504...	→	masonrothschild.eth	6.4315 Ether
0x896c226c0a47185b7f...	13762914	231 days 17 hrs ago	Gnosis Safe: Proxy Fact...	→	Contract Creation	0 Ether

Figure 14. Rarible Royalty Payment Transaction (ETH)

[10] The seller paid a total of 1.5375 ETH to cover the additional 2.5% buyer fee.
[11] 3 of the transactions were paid using ETH while the other 2 were paid using WETH, these are recorded differently on the blockchain, but as mentioned earlier, can be treated as the same for royalty purposes.

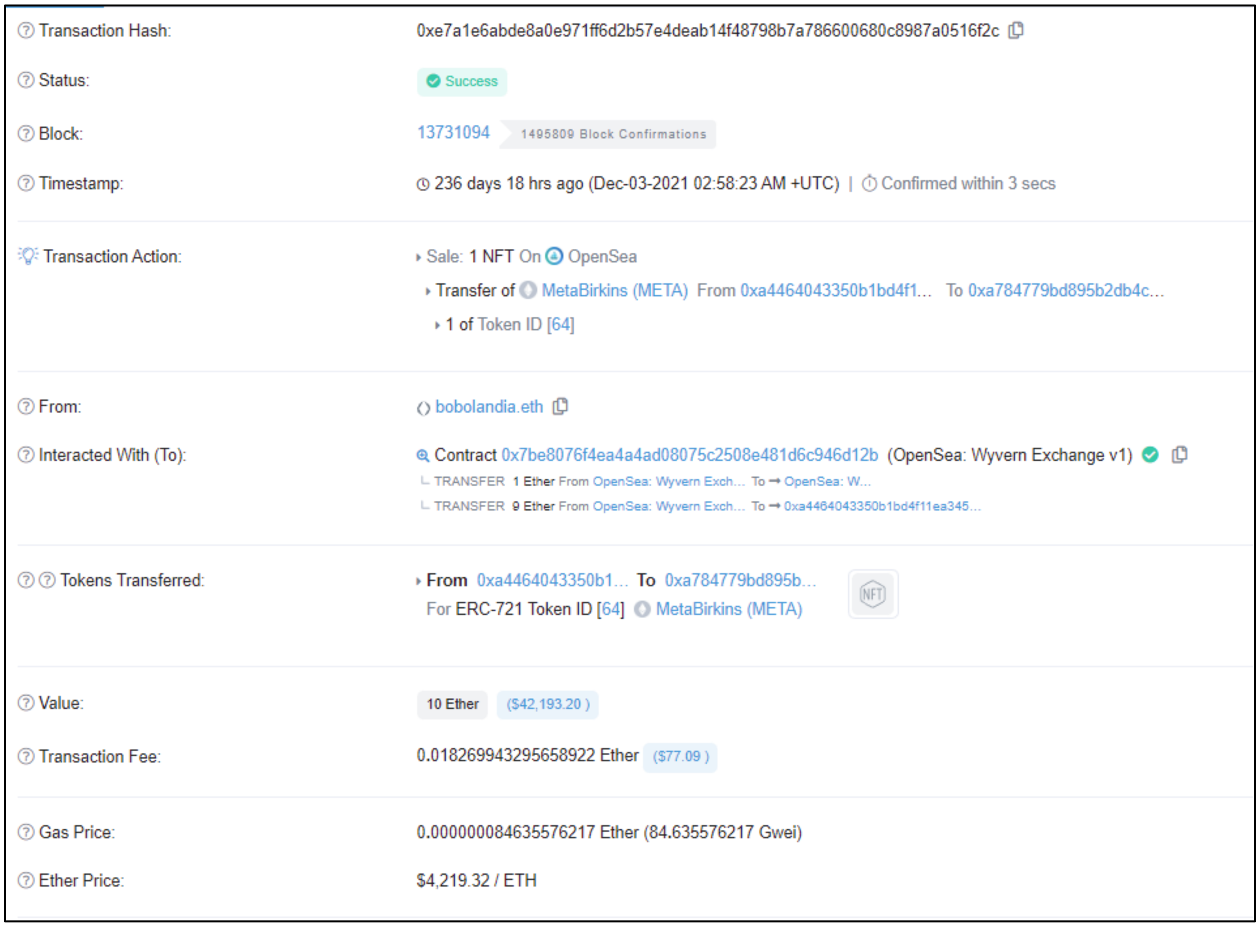


	Txn Hash	Method	Age	From		To	Quantity
👁	0x7acccb13735e05ef046...	Exec Transaction	155 days 12 hrs ago	0x0ef7e2aca9434a9504...	OUT	masonrothschild.eth	0.45
👁	0x95853a94690225519d...	0xe99a3f80	193 days 10 hrs ago	aryjawn.eth	IN	0x0ef7e2aca9434a9504...	0.225
👁	0x37d4f45a3cc686c97c8...	0xe99a3f80	195 days 15 hrs ago	0xafd1a6e363c2cbcb72d...	IN	0x0ef7e2aca9434a9504...	0.225

Figure 15. Rarible Royalty Payment Transaction (WETH)

There were 48 sales on the OpenSea Marketplace between Dec 2nd, 2021 to Jan 15th, 2022. Sales on OpenSea Marketplace do not separate out the platform fee from the royalty, instead they include a transaction that has both of these combined. The OpenSea Marketplace charges a 2.5% fee (see: https://support.opensea.io/hc/en-us/articles/1500011590241-What-are-service-and-creator-fees-) (*What Are Service and Creator Fees?*, n.d.) meaning any amount greater than 2.5% was collected on behalf of a royalty agreement. To receive a royalty on the OpenSea Marketplace, the owner of the collection sets up the royalty percentage directly on the OpenSea website and identifies the wallet that will receive the royalty payment. OpenSea Marketplace batches those transactions together to make periodic royalty payments to the address specified when the royalty fee percentage is selected. The contract owner was wallet 0xcd9d4dd3e066a77039607a82588365b75a1f4778 up until 6:13:46 AM UTC on Dec 3rd, 2021 at which time it was transferred to masonrothschild.eth. Unlike LooksRare and Rarible, there is no recording of this agreement on the blockchain and instead it needs to be inferred from the total fee collected.

Figure 16 shows an example of an OpenSea Marketplace sales transaction. In this transaction we can see that the wallet bobolandia.eth purchased MetaBirkins #64 for 10 ETH on Dec 3rd, 2021. We see 9 ETH transferred to the seller and 1 ETH transferred to OpenSea Marketplace. Based on the stated 2.5% OpenSea Marketplace fee we know the remaining 7.5% was collected for the royalty to be transferred in bulk at a later time.



Transaction Hash:	0xe7a1e6abde8a0e971ff6d2b57e4deab14f48798b7a786600680c8987a0516f2c
Status:	Success
Block:	13731094 1495809 Block Confirmations
Timestamp:	236 days 18 hrs ago (Dec-03-2021 02:58:23 AM +UTC) | Confirmed within 3 secs
Transaction Action:	Sale: 1 NFT On OpenSea Transfer of MetaBirkins (META) From 0xa4464043350b1bd4f1... To 0xa784779bd895b2db4c... 1 of Token ID [64]
From:	bobolandia.eth
Interacted With (To):	Contract 0x7be8076f4ea4a4ad08075c2508e481d6c946d12b (OpenSea: Wyvern Exchange v1) └ TRANSFER 1 Ether From OpenSea: Wyvern Exch... To → OpenSea: W... └ TRANSFER 9 Ether From OpenSea: Wyvern Exch... To → 0xa4464043350b1bd4f11ea345...
Tokens Transferred:	From 0xa4464043350b1... To 0xa784779bd895b... For ERC-721 Token ID [64] MetaBirkins (META)
Value:	10 Ether ($42,193.20)
Transaction Fee:	0.018269943295658922 Ether ($77.09)
Gas Price:	0.000000084635576217 Ether (84.635576217 Gwei)
Ether Price:	$4,219.32 / ETH

Figure 16. Sample Sale on OpenSea Marketplace

In looking at all 48 transactions that occurred on the OpenSea Marketplace, we can see that the Royalty was reduced to 0% sometime around Dec 4th, 2021 and increased back to 7.5% sometime around Dec 19th, 2021. As a result, there were 33 sales that had a 7.5% royalty and 15 sales that had no royalty.

In summary, of the 57 sales across all three marketplaces, 42 charged a 7.5% royalty and 15 charged no royalty. Of those 42 royalty-charging sales, Mason Rothschild, or his designee, received 17.34067 ETH ($69,029.30) as a result of royalties collected on the sales of MetaBirkins NFTs. As it stands now, sales can still be made on the LooksRare Marketplace and he will continue to receive 7.5% of any future sale on that platform based on the current settings.

Gifts

Finally, I also considered whether the masonrothschild.eth wallet received any MetaBirkins NFTs directly. This wallet was not whitelisted and did not mint any of the NFTs. However, after minting, the masonrothschild.eth wallet was given 3 MetaBirkins NFTs (all on Dec 3rd, 2021):

1. MetaBirkins #46 was given to masonrothschild.eth by cooodes.eth at 5:32 AM UTC

2. MetaBirkins #17 was given to masonrothschild.eth by wallet 0x5ee409ab7dc23936e817b7153e08d5f512ebd485[12] at 5:40 AM UTC
3. MetaBirkins #1 was given to masonrothschild.eth by wallet 0xc57112fb1872130a85ecf29877dd96042572a027

Using the most recent sale prior to each of these transfers (which was a MetaBirkins sale = 9.5 Ether for all three) the value of each is estimated to be $40,083. Meaning that on Dec 3rd, 2021, at the time of transfer, masonrothschild.eth received $120,249 worth of MetaBirkins NFTs.

Since the actual owner of each wallet involved in the MetaBirkins NFT Project is unknown, there may be other wallets controlled by Mason Rothschild that minted or received MetaBirkins NFTs that have not been considered here.

Fees

To launch the MetaBirkins NFT collection, on-chain fees were incurred by Mason Rothschild. These fees included the cost of putting the two smart contracts on the blockchain, adding the whitelist to the blockchain, changing the IPFS location for the collection, and destroying the minting contract after minting had concluded. To find these transactions I extracted all the transactions that used either the MetaBirkins smart contract or the MetaBirkins minting smart contract and filtered the transactions down to any initiated by either the masonrothschild.eth wallet or wallet 0xcd9d4DD3e066A77039607A82588365B75a1F4778 (i.e. the wallet that created the contracts).

Transactions related to the MetaBirkins smart contract included[13]:

- 1 Transaction for the Creation of the Smart contract
- 1 Transaction establishing the Minting Contract permissions
- 6 Transactions related to setting the base URI (IPFS Setting)
- 1 Transaction transferring ownership of the smart contract

These 9 transactions totaled 0.309950541 ETH in fees which, based on the closing price of ETH at the day of each transaction, totaled $1,392.85.

Transactions related to the Minting contract included:

- 1 Transaction for the Creation of the Smart contract
- 7 Transactions related to adding wallets to the Whitelist
- 1 Transaction related to the destruction of the smart contract

These 9 transactions totaled 0.506722321 ETH in fees which, based on the closing price of ETH at the day of each transaction, totaled $2,276.15.

In total, $3,669.00 in fees were paid to roll out and maintain the MetaBirkins NFT collection.

[12] This wallet was initially funded with wallet 0x865a98cb3ca1a22de8a5840dd99a8e98c9e5172d on Dec 2nd, 2021. That wallet was initially funded from wallet cooodes.eth also on Dec 2nd, 2021.
[13] There were 5 transactions linked to masonrothschild.eth covering the transfer and granting permissions to marketplaces related to the ownership of specific MetaBirkins NFTs that were not considered as part of the cost of rolling out the MetaBirkins NFT Project.

In conclusion, based on the transaction level details as recorded on the blockchain, Mason Rothschild, the owner of the MetaBirkins NFT Project earned at least the following:

- Minting Revenue = 10 ETH ($45,143.60)
- Royalty Revenue = 17.34067 ETH ($69,029.30)
- MetaBirkins NFT transfers = 28.5 ETH ($120,249)
- Fees Paid: -0.816672862 ETH (-$3,669.00)

4. Have MetaBirkins NFT holders received any other benefits?

Yes. Mason Rothschild launched a new NFT collection called I Like You You're Weird (ILYYW). It is common for existing NFT holders to receive benefits when the creator launches something new. As a result, I looked at whether the MetaBirkins NFT holders received any special benefits when ILYYW was launched. To answer this I examined both the ILYYW smart contract code (found here: https://etherscan.io/address/0x3f93d710ff8c449ea300ad69412e7b8f289a2954#code) as well as the transaction history of the ILYYW NFT.

The ILYYW smart contract defines 4 settings for the minting process. These settings are Closed, Reserved, Weird_list and Public. When the status is set to Closed then no minting can occur. When the status is Reserved then only the contract owner can mint. When the status is Weird_List then the wallet must have proof that they are part of that list. And when the status is Public then anyone is allowed to mint.

Shortly after the creation of the contract, the minting status was set to Reserved and 3 batches of reserved minting occurred which generated ILYYW #1 through #125. Each of these NFTs was given to the wallet that launched the collection. Several hours later the status was changed to Weird_List and the minting by individual wallets started with #126 being minted on Mar 8th, 2022 at 12:04 AM UTC. The collection sold out with ILYYW #10000 when it was minted on Mar 9th, 2022 at 1:09 AM UTC.

While many MetaBirkins NFT holders minted during the Weird_List minting, unlike the MetaBirkins, the list was not loaded to the blockchain. Because this list was managed off-chain I cannot see direct evidence that the MetaBirkins NFT holders received preferential treatment for getting onto the list. But given the large number of MetaBirkins NFT holders that also minted ILYYW NFTs there is evidence that they were at least made aware of this offering before launch.

Furthermore, on Mar 11th, 2022 the ILYYW Deployer (the account that created the ILYYW contract) distributed the first 125 ILYYW to a set of wallets through a process commonly referred to as an airdrop. An airdrop means that someone distributes cryptocurrency or other tokens (in this case ILYYW NFTs) to wallet for free. The cost of these transactions is paid for by the wallet distributing these and the wallet owners do not need to do anything to receive these. ILYYW #1 was sent to the wallet that owned the most MetaBirkins, #2 to the owner of the second most # of MetaBirkins and so forth. In other words, 84 holders of MetaBirkins NFTs received a free low-number ILYYW NFT 2 days after the ILYYW line sold out and the order gave preference to those holding multiple MetaBirkins NFTs.

5. Did the MetaBirkins NFTs differ from the BabyBirkin NFT?

Yes. Mason Rothschild partnered with the basic.space marketplace to sell the BabyBirkin NFT in May 2021. The singular BabyBirkin NFT sold for $23,500 and based on the blockchain transaction[14] that was recorded later, 65% of that sale went to Rothschild. At the time, basic.space was just launching its NFT marketplace and the BabyBirkin was one of the first NFT sales from that marketplace.

There are two major differences between the BabyBirkin NFT and the MetaBirkins NFTs. First, the BabyBirkin NFT is a short movie file while the MetaBirkins NFTs are .png files (2D still image). In general, when considering other utilities of an NFT as discussed in Section 11, it is easier to work with a 2D image than it is to work with a movie file (consider putting a 2D image on a T-shirt versus a movie on a T-shirt, the latter just cannot be done). Second, the BabyBirkin was generated on the Flow blockchain while the MetaBirkins is in the Ethereum blockchain. Flow is a much smaller blockchain and has fewer industry partnerships and is not supported by the OpenSea Marketplace (i.e. NFTs on Flow blockchain cannot be sold on OpenSea Marketplace).

6. Did the MetaBirkins NFTs differ from ILYYW NFTs?

In general, the MetaBirkins NFT collection has much more similarity with the ILYYW NFT collection than it does with the BabyBirkin NFT. I Like You You're Weird (ILYYW) launched in March 2022 on the OpenSea Marketplace and also on the Ethereum blockchain, like the MetaBirkins NFTs. The main difference is unlike MetaBirkins which had 100 unique NFTs, ILYYW was a collection of 10,000 NFTs.

The images of both collections are .png files (2D still image). Both also have their images stored on IPFS instead of on-chain allowing for future changes to be made.

Based on the file type of the image and the use of the same storage solution, it is possible that anything being done with the ILYYW collection could also be done with the MetaBirkins NFT collection.

7. Can more "MetaBirkins" NFTs be minted?

Yes. To answer this question, I examined the smart contract for the MetaBirkins NFT collection. As the contract stands today, no more MetaBirkins NFTs can be minted. This is because the minting role is connected to a smart contract that has been destroyed. However, the contract allows the minter role to be granted (by the owner of the collection, currently masonrothschild.eth) and this could be updated to point to a new minting contract that could allow for more than the current 100 MetaBirkins NFTs to be minted. While some smart contracts explicitly define the max number of NFTs, this contract has no such definition. Therefore, there is no limit to the number of MetaBirkins NFTs that could be minted.

There is nothing unusual about this process and is actually the process used to mint the first 100 MetaBirkins NFTs. First the master contract was created. Second, a minting contract was created. Third, the master contract was pointed to the minting contract. Fourth, the minting contract was used to generate 100. Fifth, the minting contract was destroyed. Steps two through five can be repeated with a new minting contract that could allow for any number of new MetaBirkins NFTs to be minted.

[14] https://flowscan.org/transaction/ea740748af52bcd59010740dfd778134b78ea77856365118a5c28a89665be099.

8. Can the "MetaBirkins" name be changed?

No. From the blockchain perspective, the MetaBirkins name cannot be changed. This name is set up when the initial contract is created and cannot be changed once it has been put on the blockchain. Due to the immutable nature of the blockchain, the original contract, which specifies the name (MetaBirkins) and token (META), cannot be destroyed or altered.

9. Can MetaBirkins NFTs be destroyed (i.e. burned)? Altered? Clawed back?

Burning a token means transferring the token to a non-existing wallet so that it non-accessible by anyone after the burn (i.e. a method of destroying the token). While the MetaBirkins smart contract has a burn function, only the current owner (not the contract owner) of the NFT can burn the NFT. The NFT would still exist, but the owner would be a non-existent wallet.

Clawing back refers to a process by which you force the NFT to be transferred to another wallet. In examining the MetaBirkins smart contract, there is no way through the contract to force an ownership transfer.

Alteration means changing some aspect of the NFT. Each MetaBirkins NFT has two characteristics that make it appear related to Hermès handbags. First is the name MetaBirkins which I have already discussed cannot be changed. Second is the image itself which can be changed. This image is stored on a decentralized repository called IPFS (InterPlanetary File System) which is a decentralized repository to store files. The contract points to the current repository and retrieves images from that repository. When each MetaBirkins was minted, it pointed to a repository that had the veiled object image (see Figure 2) associated with each one. So no matter which one you looked at, they all looked the same. The “unveiling” event changed the IPFS repository to a new folder with each bag getting a unique image (see Figure 1). A new repository could be uploaded and the pointer from the contract pointing to the new repository. While the old repository (with the current images) would still exist on IPFS, it would not by default be found with the MetaBirkins NFT.

10. Are there any other remedies?

The smart contract does allow for the transfer of ownership of the MetaBirkins contract (as was done on Dec 3rd, 2021 when the contract ownership was transferred to masonrothschild.eth). This would ensure that 1) the generation of new MetaBirkins NFTs using this same contract and 2) royalties on any future sales are controlled by the new owner. Alternatively, if the admin role was redirected to 0x0 then it would prevent any future changes to the contract.

11. How can the MetaBirkins NFTs be used?

Could they be used in the Metaverse?

Yes. Unlike the original BabyBirkin NFT, which was released as a movie file, MetaBirkins NFTs are currently linked to a .png image file. As such, any site that can pull image information from the Ethereum blockchain, that is equipped to work with a .png file, could integrate the MetaBirkins NFTs into their Metaverse.

Because the MetaBirkins NFTs are stored as .png files, it makes removing the background easy. Figure 17 shows the removal of the background on the free tool available through the site http://remove.bg. To generate this image, I simply uploaded the .png file to the site and the pedestal and background were removed automatically. The checkered pattern in the back indicates that the area is transparent which could allow for a more seamless placement of the MetaBirkins NFT. While some NFT projects restrict the use of the images through wording in the smart contract, in reviewing the smart contract I found no restrictions on use defined.

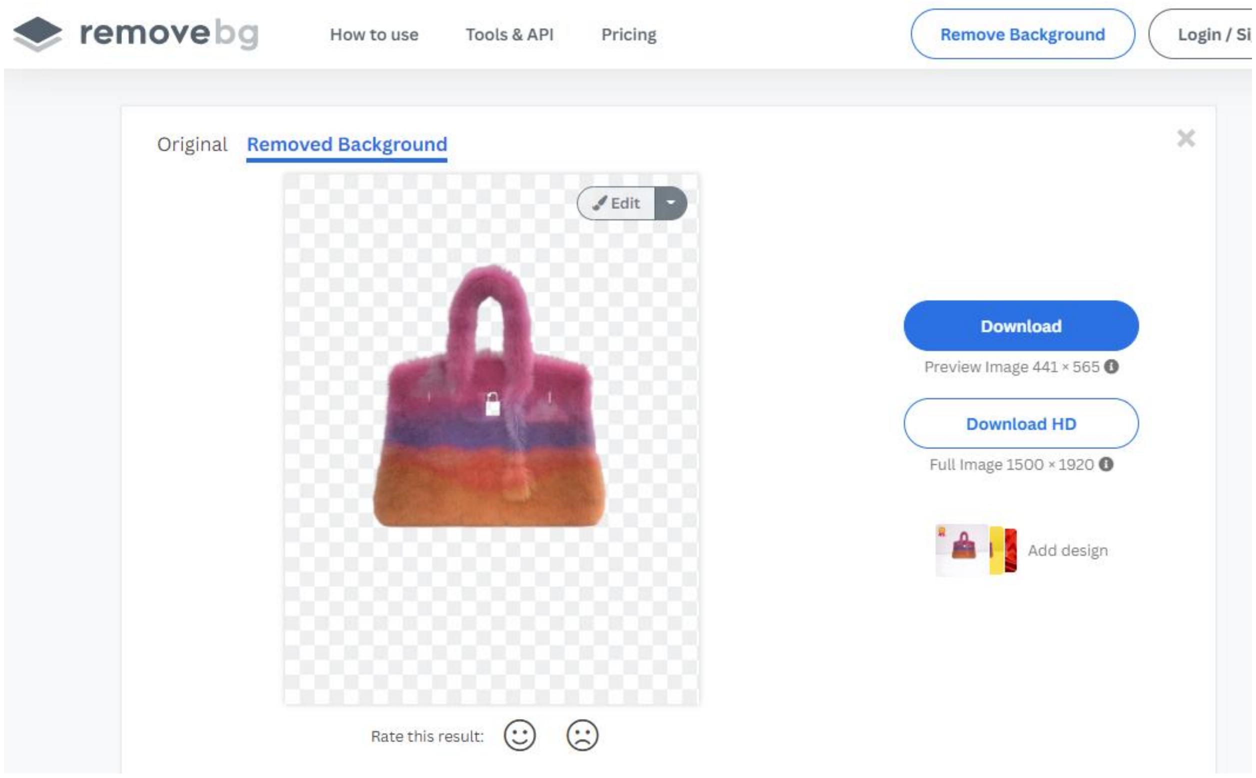
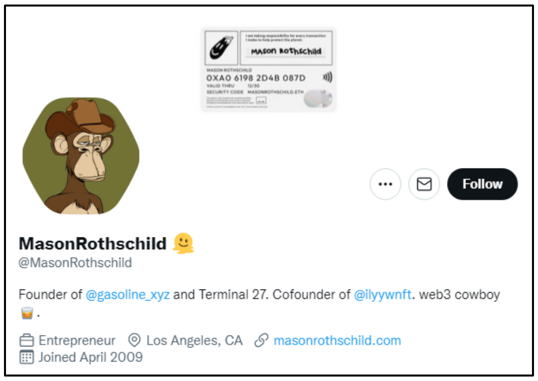


Figure 17. Easy removal of background

Could they be used on social media platforms (i.e. as an avatar or profile pic)?



MasonRothschild
@MasonRothschild

Founder of @gasoline_xyz and Terminal 27. Cofounder of @ilyywnft. web3 cowboy

Entrepreneur Los Angeles, CA masonrothschild.com
Joined April 2009

Figure 18. Mason Rothschild Twitter Profile

Yes. We are seeing social media platforms recognizing that people want to associate their account with an NFT project. For example, Twitter partnered with OpenSea Marketplace to allow Twitter users to link their wallet to their Twitter profile and select the NFT they want to use as their profile picture. Twitter went one step further and changed the shape of those NFT driven profile pictures so anyone looking at that picture would know that that Twitter user owned that NFT. Figure 18 shows Mason Rothschild's Twitter account. Notice that the image is shaped in a hexagon and not a circle like other accounts. This indicates that Mason Rothschild a) paid to subscribe to Twitter Blue (a premium service), b) linked his digital wallet to his Twitter profile, and c) selected an NFT that he owns. This NFT is Ape #215 from the Bored Ape Yacht Club NFT collection which Mason Rothschild purchased on June 21st, 2022 along with an Otherdeed NFT for 90 Eth valued at $101,223.90.

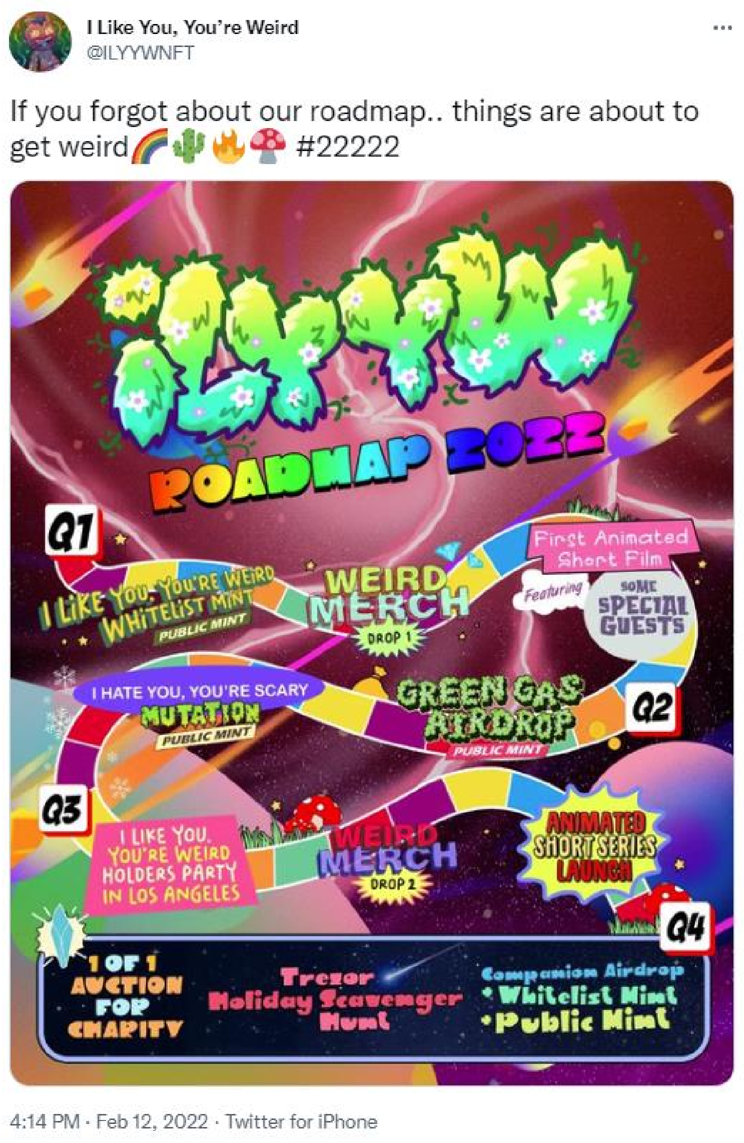
Currently the MetaBirkins NFTs cannot be used for this purpose because this functionality is a collaboration between Twitter and OpenSea Marketplace, and OpenSea Marketplace has removed the

MetaBirkins NFTs from its platform, owners of the MetaBirkins NFTs are not able to link their MetaBirkins image to their Twitter profile. However, should Twitter pull the NFT information from another platform, or directly from the blockchain, or OpenSea Marketplace reinstates the MetaBirkins NFTs then this could change.

Based on this information, Mason Rothschild is well aware that NFTs are being used on social media. He himself is paying for an extra service to be able to link one of his owned NFTs to his Twitter profile.

Could the ILYYW roadmap be implemented for MetaBirkins NFTs?

The I Like You You're Weird (ILYYW) roadmap is shown in Figure 19 and is pulled from the @ILYYWNFT Twitter feed.



I Like You, You're Weird
@ILYYWNFT

If you forgot about our roadmap.. things are about to get weird🌈🌵🔥🍄 #22222

4:14 PM · Feb 12, 2022 · Twitter for iPhone

Figure 19. ILYYW Roadmap

From this roadmap we see the following types of activities occurring:

Weird Merch

While it is unclear whether this is going to be airdropped virtual items or physical items, in either case, the design of ILYYW and MetaBirkins NFTs both using .png files suggests that whatever could be done with ILYYW NFTs could also be done with MetaBirkins NFTs.

The ILYYW collection has teamed up with Blankos Block Party (a multiplayer game by Mythical Games) to create in-game characters based on the ILYYW line of NFTs. This video, by Blankos, shows the 3D quality of the ILYYW NFTs in the game: https://www.youtube.com/watch?v=4dVAiNGDpdc.

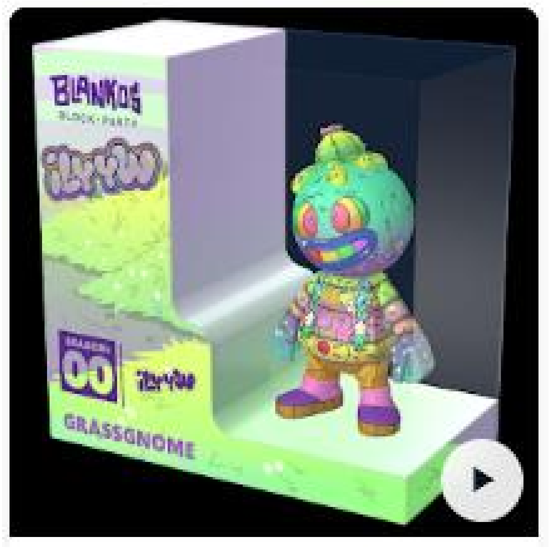


Figure 20. ILYYW X Blankos Collaboration

Accessorizing these characters with a MetaBirkins handbag is certainly something that could be easily done in a future release.

Short Films

Again, the technology behind the MetaBirkins NFTs and ILYYW NFTs is so similar that short films are reasonable to expect using either collection.

Airdrops

We have already seen the MetaBirkins NFT holders be the recipient of airdrops – as mentioned earlier, MetaBirkins NFTs holders received an ILYYW NFT through an airdrop. Airdropping new versions of MetaBirkins NFTs (3D version compatible with a Metaverse for example) is something anyone could readily do with Rothschild, as the owner of the collection, could replace the .png files with a new 3D version. Earlier we saw the image connected to the MetaBirkins go from a movie file, changed to a .jpg file, and eventually changed to the .png file. So to replace the .png with a new format is functionality is clearly available to Rothschild.

12. Could Rothschild make MetaBirkins NFTs Wearable?

While the idea of spending money for wearables that can only be used in a virtual world may seem strange to some, consumers already spend billions on digital items in games, so many see the Metaverse following this trend. While the current MetaBirkins image is a .png file, since this information is not stored on-chain (only the pointer to the image is stored and that can be changed), it is a simple process to replace this file with any other file (movie, 3D image format, music, etc.).

Luxury brands are moving into the NFT/Metaverse wearables space including:

1. Gucci partnering with 10KFT bring Gucci fashion into a virtual world (Nast, 2022)
2. Gucci and Roblox launching Gucci Town (*METAVERSE & NFT | GUCCI® VAULT US*, n.d.)
3. Dolce&Gabbana releasing a $300,000 virtual tiara (Moss, n.d.)
    a. Includes 1 custom digital recreation in open metaverse of choice
4. Dolce&Gabbana, as part of the first Metaverse Fashion Week, unveils luxury wearables for Decentraland (*Dolce&Gabbana Enters the Metaverse*, n.d.)
5. Louis Vuitton launching a virtual world where characters can be customized in Louis Vuitton fashion wear (*5 Brands Already Boldly Embracing the Metaverse*, n.d.)
6. Balenciaga partnering with Fortnight for avatar skins and accessories

Luxury brands are not the only ones moving into this space:

1. Coca Cola partnered with Tafi to create Coca Cola themed wearables for Decentraland (*5 Brands Already Boldly Embracing the Metaverse*, n.d.)
2. Nike purchased RTKFT to launch virtual Nike sneakers (Team, n.d.)

For example, I can already buy branded wearables in various Metaverses, as mentioned above. I may want to equip my avatar with the Coca-Cola bubble jacket in Decentraland or wear my Nike sneakers in Roblox.

Each of the initiatives above highlight Metaverse wearables functionality that can be done. As the owner of the MetaBirkins collection, Mason Rothschild could pursue collaborations similar to these initiatives using the MetaBirkins NFT collection. As discussed above, Mason Rothschild could replace the current .png files with new types of files for existing owners simply by pointing the MetaBirkins NFTs to a new digital file. He also could airdrop new wearable versions of MetaBirkins NFTs to owners of existing MetaBirkins NFTs.

[illegible]

Dated: August 4, 2022

____
Kevin D. Mentzer

Appendix A – Kevin D Mentzer Resume

Kevin D Mentzer, PhD

PERSONAL SUMMARY

Kevin is the Trustee Professor of Data Science at Nichols College where he is responsible for integrating technology and data literacy across the entire undergraduate student body as well as developing new majors including Data Science. His area of expertise lies in the intersection of technology, education, and pedagogy. His ongoing series of teaching cases, all with undergraduate student co-authors, is aimed at making data science accessible to an undergraduate population. He frequently publishes and presents on the topics of blockchain, NFTs, social network analysis, political discourse, and the use of social media in online communities. He is the co-founder of the annual Analytics without Borders conference as well as serving as a minitrack chair at HICSS and assistant papers chair at EDSIGCON.

EDUCATION

BENTLEY UNIVERSITY, Waltham MA
PhD in Business with a concentration in Business Analytics - 2016

- Dissertation Title: Essays on Networks of Influence – Discovering Insight through Social Network Analysis
- Dissertation Chair: Dominique M. Haughton, Professor of Mathematical Sciences, Bentley University

BENTLEY UNIVERSITY, Waltham MA
M.S in Information Technology – 2004, Highest Honors

BRYANT UNIVERSITY, Smithfield RI
B.S. in Business Administration - 1991, Cum Laude

Certified Blockchain Solution Architect (2018-2020)

TEACHING EXPERIENCE

Nichols College, Trustee Professor of Data Science Jan 2022-Present

- Responsible for designing and implementing Data Science undergraduate curriculum.
    - Curriculum aligned with ACM Data Science Model Curriculum
- Responsible for designing and implementing Data Literacy curriculum for entire undergraduate student body.
- Responsible for designing and implementing Intelligent Automation undergraduate curriculum.
- Responsible for leading digital transformation for both undergraduate and MBA programs.

Bryant University, Associate Professor June 2021-Jan 2022; Assistant Professor – Sept 2015 – May 2021

- AA304 – Managing Information in Applied Analytics
    - Tools: SAS, R, MySQL
- *AA640 – Advanced Analytics Techniques and Data Visualization*
- AA651 – Analytics Capstone in MBA Program
- MBA523 – Managing Global Information Systems
- CIS201 – Introduction to Information Systems and Analytics

- CIS470 – Managing Global Information Systems
- ISA221 – Intro to Java Programming
- *ISA310 – Data Visualization*
- *ISA330 – Programming (Python) for Analytics (replaced with course below)*
- *ISA330 – Programming for Data Science*
- ISA4xx – Directed Study in Analytics
    - Python, R
- *ISA4xx – Special Topics in Blockchain Technology*
    - Solidity
- Independent Studies
    - Blockchain and the NFT Market – Spring 2021
    - Implementing NFTs for Nichols College – Spring 2022

(courses in italics were proposed, designed, and implemented by Kevin)

Bryant University, Lecturer – Jan 2015 – May 2015

Bentley University, Adjunct Instructor – Fall 2013, 2014

- GB213 – Business Statistics
- GB310 – Business Processes & Systems
    - Tools: Process Modeler, SAP
- IPM755 – Responsible Technology: Business and Public Policy

PUBLICATIONS/PRESENTATIONS

Journal Articles

Mentzer, K., Price, J., Powers, E., Lavrenchuk, N. (Forthcoming - 2022) Examining the Hype behind the Blockchain NFT Market. *Issues in Information Systems* 23(x)

Mentzer, K, Galante, Z., Rojek, B., Cardarelli, R. (2021) March Madness: Predicting the winner of Locura de Marzo. *Issues in Information Systems* 22.4 (2021)

Mentzer, K., Frydenberg, M., & Yates, D. J. (2020). Teaching Applications and Implications of Blockchain via Project-Based Learning: A Case Study. *Information Systems Education Journal*, *18*(6), 57-85.

Mentzer, Kevin, Ying Wang, and Dominique Haughton. "Interlocking Boards and the Corporate Elite: A 20-Year Analysis of the S&P 500." *Issues in Information Systems* 21.4 (2020).

Mentzer, K., Fedorowicz, J., Williams, C (2019). Police Use of Social Media: Comparing Classification Methods. *Issues in Information Systems* 20(3)

Phan, L. Li, S., Mentzer, K. (2019) Blockchain Technology and the Current Discussion on Fraud. *Issues in Information Systems* 20(4)

Williams, C., Kavanaugh, A., Fedorowicz, J., Thatcher, J., Xu, J., Mentzer, K. (2018). Leveraging Social Media to Achieve a Community Policing Agenda. *Government Information Quarterly*

Prichard, J., Mentzer, K. (2017). An Analysis of App Privacy Statements. *Issues in Information Systems*, *18*(4).

Markus, M., Mentzer, K. (2014). Foresight for a Responsible Future with ICT. *Information Systems Frontiers, 16* (3), 353-368.

Books

Haughton, D. M., McLaughlin, M., Mentzer, K., Zhang, C. (2015). Movie Analytics A Hollywood Introduction to Big Data, Springer

Book Chapters

Haughton, D. M., McLaughlin, M., Mentzer, K., Zhang, C. (2019). Movie Analytics the Future of Film Finance, Springer (2018)

Conference Proceedings

Yates, D., Mentzer, K., Babb, J. (2022) Introduction to the Minitrack on Data Analytics, Data Mining, and Machine Learning for Social Media. HICSS 55

Mentzer, K., Galante, Z., Frydenberg M. (2021) Bracketology: Predicting Winners from Music March Madness. *EDSIGCON 2021*

Yates, D., Mentzer, K., Babb, J. (2021) Introduction to the Minitrack on Data Analytics, Data Mining, and Machine Learning for Social Media. HICSS 54

Haughton, D., Mentzer, K., Yates, D. (2020) Introduction to the Minitrack on Data Analytics, Data Mining, and Machine Learning for Social Media. HICSS 53

Mentzer, K., Fallon, K., Prichard, J., Yates. D. (2020) Measuring and Unpacking Affective Polarization on Twitter: The Role of Party and Gender in the 2018 Senate Races. HICSS 53

Mentzer, K., Frydenberg, M., Yates, D. (2019) Teaching Applications and Implications of Blockchain via Project-Based Learning: A Case Study. *EDSIGCON 2019*

Chaudhury, A, Mentzer, K. Mallick, D. (2019) Does IT Promote Collaborative Processes and Improve Learning? An Activity Theory Approach *AMCIS 2019*

Prichard, J., Mentzer K. (2017). Understanding Privacy Policy Statements in Mobile Application: A Text Analysis. *International Association for Computer Information Systems*.

Haughton, D. M., McLaughlin, M., Mentzer, K., Zhang, C. (2014). Movie Analytics: Visualization of the Co-starring Network. *IEEE, Symposium on Large Data Analysis and Visualization*.

Markus, M., Jacobson, D., Bui, Q., Lisein, O., Mentzer, K. (2013). Design of the IT Management Arrangements in the Post NPM Context. *, GT GRH Publique* .

Markus, M., Jacobson, D., Bui, Q., Mentzer, K., Lisein, O. (2013). Organizational and Institutional Arrangements for e-Government: A Preliminary Report on Contemporary IT Management Approaches in US State Governments. , 2090-2100.

Markus, M., Jacobson, D., Bui, Q., Mentzer, K., Lisein, O. (2013). IT Centralization and Enterprise-wide IT Capabilities and Outcomes: A Public Sector Study. *Proceedings of the 2013 European Conference on Information Systems*.

Presentations

Mentzer, K. (2021) “Introduction to Blockchain for Internal Auditors” Presented to Mohegan Sun, Uncasville CT.

Mentzer, K., Paul, M. (2021) “Blockchain 101 – Understanding the Current NFT Market” Presented at Analytics without Borders, Virtual.

Frydenberg, M., Mentzer K. (2020) "From Engagement to Empowerment: Project-Based Learning in Python Coding Courses." EDSIG Conference, Information Systems & Computing Academic Professionals.

Naumova, E., Li, M., Mentzer, K. (2020) “Teaching Analytics to Non-Analytics Students” Panel Participant at Analytics without Borders, Boston, MA.

Mentzer, K., Fallon, K., Prichard, J., Yates. D. (2020) The Role of Gender and Party on the Twitter Conversation in the 2018 Senate Races. Analytics without Borders, Boston, MA.

Mentzer, K, Downey, S, Sullivan, K. (2019) “Tweet Mining using Google’s Colaboratory Notebooks” Presented at RED19, Bryant University, Smithfield RI.

Mentzer, K. (2019) “Integrating the Raspberry Pi into the Classroom” Presented at the 4th Annual Analytics without Borders, Smithfield RI.

Mentzer, K., Yates, D. (2018) “Teaching Blockchain: Technology, Applications, and Implications” Panel Participant at EDSIGCON, Norfolk, VA.

Mentzer, K., Gough, M. (2018) “The Impact of Cryptokitties on the Ethereum Blockchain” Presented at Northeast Decision Sciences Institute annual meeting, Providence, RI.

Mentzer, K (2017) “Lessons Learned from Implementing Watson Analytics into the CIS201 Curriculum” Presented at REDay, Bryant University, Smithfield, RI.

Mentzer, K, Schurch (2017) “Gubernatorial Appointment Power: Using Social Network Analysis to Understand Changes in Gubernatorial Power” Presented at REDay, Bryant University, Smithfield, RI.

Mentzer, K. (2016) “Supporters, Adversaries, Peacekeepers, and Trolls: Using Text Analytics to Understand Political Messaging” Presented at SAS Day, Bryant University, Smithfield, RI.

Mentzer, K., Haughton D. (2015). “An Exploration of Power Structures Utilizing Network Analysis.” Presented at the INFORMS Annual Meeting, Philadelphia, PA.

Mentzer, K., Washington A. (2015). “Understanding Shifting Dynamics of Power in State Governments through Social Networks.” Presented at AMCIS 2015, San Juan, Puerto Rico.

Haughton, D. M., McLaughlin, M., Mentzer, K., Zhang, C. (2014). "Movie Analytics: Visualization of the Co-starring Network." Presented at the IEEE's *Symposium on Large Data Analysis and Visualization*, Paris, France.

Haughton, D. M., McLaughlin, M., Mentzer, K., Zhang, C. (2014). "Visualizing Movie Analytics: A Comparative Approach." Presented at the Bentley University's *Annual Research Showcase*, Waltham Massachusetts.

Mentzer, K. (2013). "Advanced EndNote Training." Presented at the Bentley University's *PhD Business Workshop*.

Mentzer, K. (2013). "Can We Plan for Unanticipated Consequences?" Presented at the *Bentley University Research Colloquium*.

Markus, M., Mentzer, K. (2013). "Future Search: Research about ICT and the Cultural Contradictions of Capitalism." Presented at the Academy of Management Annual Meeting's *Professional Development Workshop*, Orlando, FL.

Markus, M., Mentzer, K. (2013). "Design for a Better Future: Are Today's Sociotechnical Methods Adequate?" Presented at the NSF and University of Maryland's *Digital Societies and Sociotechnical Systems (DSST) 2013*, College Park, MD.

Markus, M., Jacobson, D., Bui, Q., Mentzer, K., Lisein, O. (2013). "IT Centralization and Enterprise-Wide Capabilities and Outcomes: A Public Sector Study"." Presented at the *21st European Conference on Information Systems*, Utrecht, The Netherlands.

Markus, M., Jacobson, D., Bui, Q., Lisein, O., Mentzer, K. (2013). "Design of the IT Management Arrangements in the Post NPM Context." Presented at the GT GRH Publique , Liege, Belgium.

Professional Experience

Energy Services Group Inc., Senior IT Consultant	Jan '10 - May '11
Blue Cod Technologies Inc., Director: Software Development	Jan '04 - Jan '10
CounterPoint Technologies Corp., Founder and Senior Consultant	Dec '94 – Jan '10
Progress Software, Inc., Consultant	Dec '93 – Dec '94
Hanover Insurance Corp., Sr Systems Analyst	May '91 – Dec '93

University Service

- University Curriculum Committee chair for the School of Business (2018-present)
- Faculty advisor to Data Science Club and Bryant Blockchain Club
    - Successfully raised $11,000 to launch Bryant Blockchain Club
- Co-Founder/Co-Leader of **Analytics Without Borders**, an Annual Conference on Applied Analytics (2014-Present)
- Co-chair of the minitrack on Data Analytics, Data Mining, and Machine Learning for Social Media **HICSS** 2018-present
- Member of Steering Committee for NSF Funded workshop **Big Data, Big Decisions**, Jan 2014
- Member of Steering Committee for Bentley University Research Council; Fall 2012 – May 2014
- Ph.D. Council Student Representative – elected by peers; Spring 2013 – Spring 2015
- Reviewer:
    - AMCIS, 2015 tracks in SIGeGov and SIGDSA (Data and Text Mining)
    - ICIS, 2021
    - Journal of Strategic Information Systems 2011-Present

- ECIS, 2013
- HICSS, eGov Track 2013

Appendix B – Timeline

Key Events (all times listed are +UTC) related to the MetaBirkins NFT market

12/2/2021

6:00:26 AM	MetaBirkins (META) Contract Created on Ethereum Blockchain by 0xcd9d4dd3e066a77039607a82588365b75a1f4778
6:08:30 AM	MetaBirkins Minting Contract Created on Ethereum Blockchain by 0xcd9d4dd3e066a77039607a82588365b75a1f4778
6:20:45 AM	First Wallet 0x278db415b3c969e789e1ec9e80e1e001a5dcee82 added to Whitelist
10:52:10PM	Image repository set to ipfs://QmRP8Bmd36gP2pgBHXTTvgMeLjuVR43TnL25VboSh1ZSeu/
10:58:36 PM	98 Wallets added to Whitelist
11:00:16 PM	#100 Wallet 0xc57112fb1872130a85ecf29877dd96042572a027 added to Whitelist
11:01:44 PM	MB #0 Minted by 0x278db415b3c969e789e1ec9e80e1e001a5dcee82
11:01:48 PM	MB #1 Minted by 0xc57112fb1872130a85ecf29877dd96042572a027
11:17:55 PM	MB #2 Minted by 0x9c097a3124495f129265c8a1f44c61be55422139

12/3/2021

12:01:06AM	Image repository set to ipfs://QmaY9LGQoicwxapUghMGmkW7E4d564YByJ77E8FyWFPTjp/
12:03:54AM	Image repository set to ipfs://QmaY9LGQoicwxapUghMGmkW7E4d564YByJ77E8FyWFPTjp/
12:16:56AM	Image repository set to ipfs://QmbC7V7sESpWAw2ZmnccY3xaP2iJwsFdXUBQqB5abQj8LN/
01:57:52AM	Image repository set toipfs://Qmc5Qi4homP5mrnnUzxbcQziZqbXP74ujtXMTq8FBR7GaF/

At this point all bags have the Veiled Object image.
For example: Image for bag #64
{
- name: "65",
- image: "ipfs://QmRWq9xwyfAHoUqXcB7Fb7PxCDmby8HqKrksnGNudsAthY"

}

1:03:00 AM	MB #3 Minted by 0xa244f5434a77e09f282e727aa1317517b0c944f1
1:04:24 AM	MB #4 Minted by 0xd8443486d005558ab49bc288d425612dbed6a958
1:05:35 AM	MB #5 Minted by 0x23274a30baa2d8b3b15464857acbe744d1a2b592
1:44:39 AM - 8:09:29 AM	MB #6 through MB #93 Minted
2:07:59 AM	MB Owners start granting permissions to OpenSea Marketplace (to enable selling)
02:58:23AM	MB 64 Sold on OpenSea for 10.0 ETH with 7.5% Royalty
03:10:09AM - 05:24:37AM	10 additional MBs Sold on Open Sea with range 4.5-7.5 ETH with 7.5% Royalty
05:24:37AM	MB 15 Transfer • From: 0x56caf3bc01f69b4ffe3625dbfcb563bde9270f6a • To: 0x734b955dbdec0fea40511556fa07f6b6f5a1ee7d
05:31:16AM	MB 16 Transfer • From: 0x865a98cb3ca1a22de8a5840dd99a8e98c9e5172d • To: 0xc477e5b2cb6d972dad2b2fe385c9dae42333d193 (babybirkins.eth)
05:33:51AM	MB 53 Sold for 9.5 ETH
05:39:29AM	MB 46 Transfer • From: 0xcc6d4546f57ae7d37a2acb17d07228e0d8439e6c (cooodes.eth) • To: 0xa061982d4b087d911db8399a641187df945d48d0 (masonrothschild.eth)
05:40:46AM	MB 17 Transfer • From: 0x5ee409ab7dc23936e817b7153e08d5f512ebd485 • To: 0xa061982d4b087d911db8399a641187df945d48d0 (masonrothschild.eth)
05:58:18AM	MB 17 Transfer • From: 0xa061982d4b087d911db8399a641187df945d48d0 (masonrothschild.eth) • To: 0x93d95608Bc5Fc0331ef12Bd50d6e0a333ca3fb07
06:13:46AM	MetaBirkins contract changed ownership to masonrothschild.eth
08:39:3AM	MB 88 Sold for 9.4 ETH
10:39:49AM	MB 49 Sold for 9.659 ETH
10:41:22AM	MB 49 Transfer • From: 0xe5859cbc7a5c954d33480e67266c2bbc919a966e • To: 0x787a296f1b7646453cba1df483801e3dbba29396 (anamika.eth)
02:00:42PM	MB 52 Sold for 9.5 ETH
03:29:05PM	MB 28 Transfer • From: 0x4cb4c2f595e737ffa135773cdda8ee482ba52a89 (latevault.eth) • To: 0x261e6e7aaacfcde75535fe96943f35bcd86b84c7 (latefx.eth)
05:07:39PM	MB 1 Transfer • From: 0xc57112fb1872130a85ecf29877dd96042572a027 • To: 0xa061982d4b087d911db8399a641187df945d48d0 (masonrothschild.eth)
05:51:34PM	MB 40 Sold for 7.85 ETH ($33,121.66)

7:01:11 PM Final MB (#99) minted by 0x3d32673ae56045dbd2704e7692d2b50b59f14534

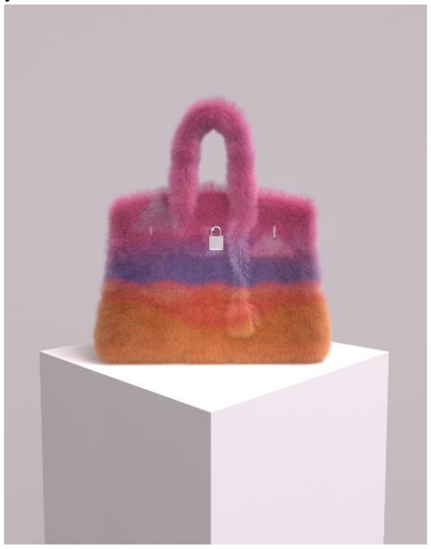
7:03:06 PM Minting Contract Destroyed with 9 ETH transferred to masonrothschild.eth and 1 ETH transferred to 0xcd9d4dd3e066a77039607a82588365b75a1f4778

07:34:04PM Images of individual bags released

Image repository set to ipfs://QmNx4bXUJP65eAcqtQnu5HSQdLcRTpWXXCrkqbw4wJbr8w/

For example: Image for bag #64
{
- name: "65",
- image: " ipfs://QmeRGrrgxP1u9muNZsxsfTvXq6HDQ442mD3GbdB9Ka1s5P/0-96.png "

}



08:32:09PM MB 33 Transfer
- From: 0x514a8bddbb1afbe1360e49d9702fb3917e66a4d4
- To: 0x2004c725dbfe6d7a4cd0c37e976c335b825a02ae

09:20:51PM MB 94 Transfer
- From: 0x55d6d25ae0b1280392f98cd30724defb1fea849b (spicyvodka.eth)
- To: 0x0cc1a222be2128686ee8a5837cd82811f58ca16e (lalaanthony.eth)

End of Day 12/3/2021: Total sales for day: 115.7589 ETH ($488,427.639)

12/4/2021 3 MBs sold (3-9.9 ETH) with 7.5% Royalty

1 MB Sold (2.5 ETH) with 0% Royalty

12/5/2021 3 MBs Sold (1-3 ETH) with 0% Royalty

12/6/2021 1 MBs Sold (4 ETH) with 0% Royalty

12/7/2021 6 MBs Sold (3.33-5 ETH) with 0% Royalty

12/8/2021 1 MB Sold (4 ETH) with 0% Royalty

12/9/2021	3 MBs Sold (4.5-8 ETH) with 0% Royalty
	Royalty on OpenSea Marketplace set back to 7.5%
	1 MB Sold (6 ETH) with 7.5% Royalty
12/11/2021 - 12/19/2021	11 MBs Sold (4-6.8 ETH) on OpenSea Marketplace with 7.5% Royalty
12/20/2021	MB Owners start granting permissions to Rarible Marketplace (to enable selling)
12/26/2021	First MB Sale on Rarible (#77) for 1.5 ETH with 7.5% Royalty going to 0x0Ef7E2aCa9434a95045b6d1D92Bc774F711C0ca8
1/10/2022	MB Owners start granting permissions to LooksRare Marketplace (to enable selling)
1/11/2022 - 1/16/2022	4 MBs Sold on Rarible with 7.5% Royalty going to: 0x0Ef7E2aCa9434a95045b6d1D92Bc774F711C0ca8 (ETH) or 0x0ef7e2aca9434a95045b6d1d92bc774f711c0ca8 (WETH)
1/15/2022	Final OpenSea Marketplace sale MD #35 for 4 ETH
1/23/2022	First sale on LooksRare (#47) for 1.5 WETH with 7.5% going directly to masonrothschild.eth
2/12/2022	Sale #95 on LooksRare for 3.5 WETH with 7.5% going directly to masonrothschild.eth
3/11/2022	Final 2 sales on LooksRare for 3.5 WETH each with 7.5% going directly to masonrothschild.eth

End of Transaction Timeline

Appendix C – MetaBirkins Sales ($ Values based on ETH closing price on date of sale)

Date Time (UTC)	Action	Buyer	Token ID	Price	Market	Royalty	Price	Currency	Value $	Royalty $	Royalty Crypto	# Txns
1/23/2022 4:00	Bid Won	0x7b724878f967f3731a0ac6a5274 5a05269b0e9db	47	1.5 WETH ($3,812.40)	LooksRare	0.075	1.5	WETH	$3,812.40	$285.93	0.1125	
2/12/2022 4:45	Bought	0xf3fd33ce765115a2d1e70acb10e8 299b9b92b8bb	95	3.5 WETH ($10,214.19)	LooksRare	0.075	3.5	WETH	$10,214.19	$766.06	0.2625	
3/11/2022 1:13	Bid Won	0xf0d6999725115e3ead3d927eb33 29d63afaec09b	17	3.5 WETH ($8,951.50)	LooksRare	0.075	3.5	WETH	$8,951.50	$671.36	0.2625	
3/11/2022 1:14	Bid Won	0xf0d6999725115e3ead3d927eb33 29d63afaec09b	31	3.5 WETH ($8,951.50)	LooksRare	0.075	3.5	WETH	$8,951.50	$671.36	0.2625	
						WETH Total	**12**		**$31,929.59**	**$2,394.72**	**0.9**	**4**
12/3/2021 2:58	Bought	0xa784779bd895b2db4c0009a746 8f090012e12ff9	64	10 ETH ($42,193.20)	OpenSea	0.075	10	ETH	$42,193.20	$3,164.49	0.75	
12/3/2021 3:10	Bought	0x78e37504ac03ce61ac5d5978592 c10b38a8dbcff	27	4.5 ETH ($18,986.94)	OpenSea	0.075	4.5	ETH	$18,986.94	$1,424.02	0.3375	
12/3/2021 3:29	Bought	0x67f3a567762f212822ccb47c348c 3ce3d9f6db7a	32	7.5 ETH ($31,644.90)	OpenSea	0.075	7.5	ETH	$31,644.90	$2,373.37	0.5625	
12/3/2021 4:08	Bought	0x9199aa7f8715121a7aa8b7d2e87 b89f05196b6ac	12	5.25 ETH ($22,151.43)	OpenSea	0.075	5.25	ETH	$22,151.43	$1,661.36	0.39375	
12/3/2021 4:46	Bought	0xde7f82ce6055b79b0d121cbf998f 318d21cc75ca	52	4.8 ETH ($20,252.74)	OpenSea	0.075	4.8	ETH	$20,252.74	$1,518.96	0.36	
12/3/2021 4:58	Bought	0xde7f82ce6055b79b0d121cbf998f 318d21cc75ca	40	5 ETH ($21,096.60)	OpenSea	0.075	5	ETH	$21,096.60	$1,582.25	0.375	

12/3/2021 5:08	Bought	0xd8891897f4b87af7a79b116ed170d976c7646fcc	50	5.2999 ETH ($22,361.97)	OpenSea	0.075	5.2999	ETH	$22,361.97	$1,677.15	0.397493	
12/3/2021 5:16	Bought	0x4cb4c2f595e737ffa135773cdda8ee482ba52a89	28	6 ETH ($25,315.92)	OpenSea	0.075	6	ETH	$25,315.92	$1,898.69	0.45	
12/3/2021 5:19	Bought	0x5edb7dc2e3fc7f4f42c0e4283dcb067252d89291	31	5 ETH ($21,096.60)	OpenSea	0.075	5	ETH	$21,096.60	$1,582.25	0.375	
12/3/2021 5:24	Bought	0x7a6397713631c3b51625ddbb11fc96e669f184b3	58	5 ETH ($21,096.60)	OpenSea	0.075	5	ETH	$21,096.60	$1,582.25	0.375	
12/3/2021 5:33	Bought	0xb5c105ed531bc6ad1c2c38372dc797d47c01bb67	53	9.5 ETH ($40,083.54)	OpenSea	0.075	9.5	ETH	$40,083.54	$3,006.27	0.7125	
12/3/2021 8:39	Bought	0xbbb677a94eda9660832e9944353dd6e814a45705	88	9.4 ETH ($39,661.61)	OpenSea	0.075	9.4	ETH	$39,661.61	$2,974.62	0.705	
12/3/2021 10:39	Bought	0xe5859cbc7a5c954d33480e67266c2bbc919a966e	49	9.659 ETH ($40,754.41)	OpenSea	0.075	9.659	ETH	$40,754.41	$3,056.58	0.724425	
12/3/2021 14:00	Bought	0x8c35933c469406c8899882f5c2119649cd5b617f	52	9.5 ETH ($40,083.54)	OpenSea	0.075	9.5	ETH	$40,083.54	$3,006.27	0.7125	
12/3/2021 17:51	Bought	0xde5fb7260d57ba0e4f7f1beab6217f5a6cedfe85	40	7.85 ETH ($33,121.66)	OpenSea	0.075	7.85	ETH	$33,121.66	$2,484.12	0.58875	
12/3/2021 20:01	Bought	0x48889a2cea22b794d7bcbfd156a404ba37b1776c	58	7 ETH ($29,535.24)	OpenSea	0.075	7	ETH	$29,535.24	$2,215.14	0.525	
12/4/2021 15:13	Bought	0x1723a88158118ad1ae03a873b2df1e8de4fb921e	12	9.9 ETH ($40,830.17)	OpenSea	0.075	9.9	ETH	$40,830.17	$3,062.26	0.7425	
12/5/2021 5:16	Bought	0x3008e2cc65c47f0c30e0555e099d80f1dd9a2a63	5	1 ETH ($4,201.09)	OpenSea	0	1	ETH	$4,201.09	$0.00	0	
12/5/2021 5:34	Bought	0x775936c4dca762d38e329930c60dc4a71b724ca1	4	3 ETH ($12,603.27)	OpenSea	0	3	ETH	$12,603.27	$0.00	0	
12/6/2021 20:56	Bought	0x6d33ad572e072b97ffea834cb7be668490afe6cd	3	4 ETH ($17,428.68)	OpenSea	0	4	ETH	$17,428.68	$0.00	0	

12/7/2021 1:38	Bought	0xa784779bd895b2db4c0009a7468f090012e12ff9	21	3.9 ETH ($16,807.83)	OpenSea	0	3.9	ETH	$16,807.83	$0.00	0	
12/7/2021 2:23	Bought	0xa784779bd895b2db4c0009a7468f090012e12ff9	11	5 ETH ($21,548.50)	OpenSea	0	5	ETH	$21,548.50	$0.00	0	
12/7/2021 3:00	Bought	0xa784779bd895b2db4c0009a7468f090012e12ff9	59	5 ETH ($21,548.50)	OpenSea	0	5	ETH	$21,548.50	$0.00	0	
12/7/2021 3:41	Bought	0xa784779bd895b2db4c0009a7468f090012e12ff9	30	3.9 ETH ($16,807.83)	OpenSea	0	3.9	ETH	$16,807.83	$0.00	0	
12/7/2021 4:30	Bought	0x78e37504ac03ce61ac5d5978592c10b38a8dbcff	28	3.33 ETH ($14,351.30)	OpenSea	0	3.33	ETH	$14,351.30	$0.00	0	
12/7/2021 23:33	Bought	0x4fcea0a8fc0df0ce0d653ece0c79e2b75a30f4d0	69	4 ETH ($17,238.80)	OpenSea	0	4	ETH	$17,238.80	$0.00	0	
12/8/2021 5:11	Bought	0x13fc16d47ce2578b5ab6a0eafc6854f62851086b	88	4 ETH ($17,759.92)	OpenSea	0	4	ETH	$17,759.92	$0.00	0	
12/9/2021 8:49	Bought	0xa784779bd895b2db4c0009a7468f090012e12ff9	63	4.5 ETH ($18,503.64)	OpenSea	0	4.5	ETH	$18,503.64	$0.00	0	
12/9/2021 20:27	Bought	0xa784779bd895b2db4c0009a7468f090012e12ff9	2	8 ETH ($32,895.36)	OpenSea	0	8	ETH	$32,895.36	$0.00	0	
12/9/2021 21:49	Bought	0x3b0f3cfbbbd9402cb95391fe6a1bc64267db4f07	75	4.95 ETH ($20,354.00)	OpenSea	0	4.95	ETH	$20,354.00	$0.00	0	
12/11/2021 3:59	Bought	0x79e087a1957fccf93649d3732f9193f0fc8c5138	73	5.7 ETH ($23,305.48)	OpenSea	0.075	5.7	ETH	$23,305.48	$1,747.91	0.4275	
12/13/2021 21:04	Bought	0xa784779bd895b2db4c0009a7468f090012e12ff9	86	6.8 ETH ($25,738.54)	OpenSea	0.075	6.8	ETH	$25,738.54	$1,930.39	0.51	
12/14/2021 20:38	Bought	0x38b9c791a6aa299eed24e60feadbf438198ee26c	61	5.8 ETH ($22,400.64)	OpenSea	0.075	5.8	ETH	$22,400.64	$1,680.05	0.435	
12/14/2021 20:44	Bought	0xa784779bd895b2db4c0009a7468f090012e12ff9	50	5.5 ETH ($21,241.99)	OpenSea	0.075	5.5	ETH	$21,241.99	$1,593.15	0.4125	

12/16/2021 0:40	Bought	0xf31f8d16601c7cc8796499354e96de0a1200ba76	34	6 ETH ($23,742.18)	OpenSea	0.075	6	ETH	$23,742.18	$1,780.66	0.45	
12/16/2021 3:38	Bought	0x61453a32c3e8cfdb70f5ec5cb9bd654a1fa46321	20	5.25 ETH ($20,774.41)	OpenSea	0.075	5.25	ETH	$20,774.41	$1,558.08	0.39375	
12/18/2021 18:42	Bought	0x694de865c83e90ebd8876f98b6a9fe5dc2182169	75	5 ETH ($19,809.80)	OpenSea	0.075	5	ETH	$19,809.80	$1,485.74	0.375	
1/15/2022 1:57	Bought	0x3f8b5e08d8d0eedb36e13c3ed6bbc7350acceaa0	35	4 ETH ($13,314.52)	OpenSea	0.075	4	ETH	$13,314.52	$998.59	0.3	
						ETH Total	**219.7889**		**$912,643.35**	**$51,044.60**	**12.39067**	**38**
12/3/2021 4:04	Bid Won	0x56f8503ebdee6214f5616bb79537fff806cc4c52	56	4.9 WETH ($20,674.67)	OpenSea	0.075	4.9	WETH	$20,674.67	$1,550.60	0.3675	
12/4/2021 1:16	Bid Won	0x78e37504ac03ce61ac5d5978592c10b38a8dbcff	54	5.5 WETH ($22,683.43)	OpenSea	0.075	5.5	WETH	$22,683.43	$1,701.26	0.4125	
12/4/2021 21:49	Bid Won	0xa784779bd895b2db4c0009a7468f090012e12ff9	48	3 WETH ($12,372.78)	OpenSea	0.075	3	WETH	$12,372.78	$927.96	0.225	
12/4/2021 23:00	Bid Won	0xa784779bd895b2db4c0009a7468f090012e12ff9	37	2.5 WETH ($10,310.65)	OpenSea	0	2.5	WETH	$10,310.65	$0.00	0	
12/5/2021 5:09	Bid Won	0x78e37504ac03ce61ac5d5978592c10b38a8dbcff	39	2 WETH ($8,402.18)	OpenSea	0	2	WETH	$8,402.18	$0.00	0	
12/9/2021 21:55	Bid Won	0xa784779bd895b2db4c0009a7468f090012e12ff9	22	6 WETH ($24,671.52)	OpenSea	0.075	6	WETH	$24,671.52	$1,850.36	0.45	
12/14/2021 20:30	Bid Won	0xa784779bd895b2db4c0009a7468f090012e12ff9	38	6.5 WETH ($25,104.17)	OpenSea	0.075	6.5	WETH	$25,104.17	$1,882.81	0.4875	
12/15/2021 21:50	Bid Won	0xaa8e798172b5ebc0ad29099ec971fb78e6df0166	36	4.5 WETH ($18,090.86)	OpenSea	0.075	4.5	WETH	$18,090.86	$1,356.81	0.3375	
12/18/2021 22:54	Bid Won	0x1350a306b27b1f650423757e0ca50d65aa4f2093	82	4 WETH ($15,847.84)	OpenSea	0.075	4	WETH	$15,847.84	$1,188.59	0.3	

12/19/2021 23:17	Bid Won	0xf63a4d80d28f8c304892b989c6ad315cb4951d5f	18	4.5 WETH ($17,660.48)	OpenSea	0.075	4.5	WETH	$17,660.48	$1,324.54	0.3375	
						WETH Total	**43.4**		**$175,818.58**	**$11,782.93**	**2.9175**	**10**
12/26/2021 17:15	Bid Won	0xa6c80dc48783eb9d3e3ca4ea1e1dad61e5adeb2a	77	1.5 ETH ($6,095.16)	Rarible	0.075	1.5	ETH	$6,095.16	$457.14	0.1125	
1/11/2022 17:29	Bid Won	0x901b8708f9516c84d777a74f142e44b89f437b93	83	3.8 ETH ($12,313.22)	Rarible	0.075	3.8	ETH	$12,313.22	$923.49	0.285	
1/14/2022 6:02	Bid Won	0xe5fb12703a7c83859d84cba18958484c5a1309b8	67	3.8 ETH ($12,576.71)	Rarible	0.075	3.8	ETH	$12,576.71	$943.25	0.285	
						ETH Total	**9.1**		**$30,985.09**	**$2,323.88**	**0.6825**	**3**
1/13/2022 23:02	Bought	0xafd1a6e363c2cbcb72dbb583aa3084a10a13b800	15	3 WETH ($9,726.78)	Rarible	0.075	3	WETH	$9,726.78	$729.51	0.225	
1/16/2022 4:12	Bought	0x7d0c87c3574cb72c96a128666ffa27cdecc8aaef	0	3 WETH ($10,048.83)	Rarible	0.075	3	WETH	$10,048.83	$753.66	0.225	
						WETH Total	**6**		**$19,775.61**	**$1,483.17**	**0.45**	**2**
						Grand Total	**290.2889**	**ETH/WETH**	**$1,171,152.22**	**$69,029.30**	**17.34067**	**57**

Appendix D - Smart Contract Code

MetaBirkins.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.2;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol";
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";
import "@openzeppelin/contracts/access/AccessControl.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/Counters.sol";

contract MetaBirkins is ERC721, Ownable, ERC721Enumerable, ERC721URIStorage, AccessControl {
    using Counters for Counters.Counter;

    bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");

    string _tokenBaseURI;

    Counters.Counter _tokenIdCounter;
    mapping(uint256 => bool) _upgradedTokens;
    mapping(uint256 => string) _tokenURIs;

    constructor() ERC721("MetaBirkins", "META") {
        _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
        _setupRole(MINTER_ROLE, msg.sender);
    }

    function _baseURI() internal view override returns (string memory) {
        return _tokenBaseURI;
    }

    function safeMint(address to) public onlyRole(MINTER_ROLE) {
        _safeMint(to, _tokenIdCounter.current());
        _tokenIdCounter.increment();
    }

    function setBaseURI(string calldata baseURI) public onlyRole(DEFAULT_ADMIN_ROLE) {
        _tokenBaseURI = baseURI;
    }
    // The following functions are overrides required by Solidity.

    function _beforeTokenTransfer(address from, address to, uint256 tokenId)
        internal
        override(ERC721, ERC721Enumerable)
    {
        super._beforeTokenTransfer(from, to, tokenId);
    }

    function _burn(uint256 tokenId) internal override(ERC721, ERC721URIStorage) {
        super._burn(tokenId);
    }

    function tokenURI(uint256 tokenId)
        public
        view
        override(ERC721, ERC721URIStorage)
        returns (string memory)
    {
        return super.tokenURI(tokenId);
    }

    function supportsInterface(bytes4 interfaceId)
        public
        view
        override(ERC721, ERC721Enumerable, AccessControl)
        returns (bool)
    {
        return super.supportsInterface(interfaceId);
    }
}
```

Counter.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

/**
 * @title Counters
 * @author Matt Condon (@shrugs)
 * @dev Provides counters that can only be incremented, decremented or reset. This can be used e.g. to track the number
 * of elements in a mapping, issuing ERC721 ids, or counting request ids.
 *
 * Include with `using Counters for Counters.Counter;`
 */
library Counters {
    struct Counter {
        // This variable should never be directly accessed by users of the library: interactions must be restricted to
        // the library's function. As of Solidity v0.5.2, this cannot be enforced, though there is a proposal to add
        // this feature: see https://github.com/ethereum/solidity/issues/4637
        uint256 _value; // default: 0
    }

    function current(Counter storage counter) internal view returns (uint256) {
        return counter._value;
    }

    function increment(Counter storage counter) internal {
        unchecked {
            counter._value += 1;
        }
    }

    function decrement(Counter storage counter) internal {
        uint256 value = counter._value;
        require(value > 0, "Counter: decrement overflow");
        unchecked {
            counter._value = value - 1;
        }
    }

    function reset(Counter storage counter) internal {
        counter._value = 0;
    }
}
```

Ownable.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "../utils/Context.sol";

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
abstract contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor() {
        _setOwner(_msgSender());
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view virtual returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public virtual onlyOwner {
        _setOwner(address(0));
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        _setOwner(newOwner);
    }

    function _setOwner(address newOwner) private {
        address oldOwner = _owner;
        _owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);
    }
}
```

AccessControl.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "./IAccessControl.sol";
import "../utils/Context.sol";
import "../utils/Strings.sol";
import "../utils/introspection/ERC165.sol";

/**
 * @dev Contract module that allows children to implement role-based access
 * control mechanisms. This is a lightweight version that doesn't allow enumerating role
 * members except through off-chain means by accessing the contract event logs. Some
 * applications may benefit from on-chain enumerability, for those cases see
 * {AccessControlEnumerable}.
 *
 * Roles are referred to by their `bytes32` identifier. These should be exposed
 * in the external API and be unique. The best way to achieve this is by
 * using `public constant` hash digests:
 *
 * ```
 * bytes32 public constant MY_ROLE = keccak256("MY_ROLE");
 * ```
 *
 * Roles can be used to represent a set of permissions. To restrict access to a
 * function call, use {hasRole}:
 *
 * ```
 * function foo() public {
 *     require(hasRole(MY_ROLE, msg.sender));
 *     ...
 * }
 * ```
 *
 * Roles can be granted and revoked dynamically via the {grantRole} and
 * {revokeRole} functions. Each role has an associated admin role, and only
 * accounts that have a role's admin role can call {grantRole} and {revokeRole}.
 *
 * By default, the admin role for all roles is `DEFAULT_ADMIN_ROLE`, which means
 * that only accounts with this role will be able to grant or revoke other
 * roles. More complex role relationships can be created by using
 * {_setRoleAdmin}.
 *
```

```
 * WARNING: The `DEFAULT_ADMIN_ROLE` is also its own admin: it has permission to
 * grant and revoke this role. Extra precautions should be taken to secure
 * accounts that have been granted it.
 */
abstract contract AccessControl is Context, IAccessControl, ERC165 {
    struct RoleData {
        mapping(address => bool) members;
        bytes32 adminRole;
    }

    mapping(bytes32 => RoleData) private _roles;

    bytes32 public constant DEFAULT_ADMIN_ROLE = 0x00;

    /**
     * @dev Modifier that checks that an account has a specific role. Reverts
     * with a standardized message including the required role.
     *
     * The format of the revert reason is given by the following regular expression:
     *
     *  /^AccessControl: account (0x[0-9a-f]{40}) is missing role (0x[0-9a-f]{64})$/
     *
     * _Available since v4.1._
     */
    modifier onlyRole(bytes32 role) {
        _checkRole(role, _msgSender());
        _;
    }

    /**
     * @dev See {IERC165-supportsInterface}.
     */
    function supportsInterface(bytes4 interfaceId) public view virtual override returns (bool) {
        return interfaceId == type(IAccessControl).interfaceId || super.supportsInterface(interfaceId);
    }

    /**
     * @dev Returns `true` if `account` has been granted `role`.
     */
    function hasRole(bytes32 role, address account) public view override returns (bool) {
        return _roles[role].members[account];
    }

    /**
     * @dev Revert with a standard message if `account` is missing `role`.
     *
     * The format of the revert reason is given by the following regular expression:
     *
     *  /^AccessControl: account (0x[0-9a-f]{40}) is missing role (0x[0-9a-f]{64})$/
     */
    function _checkRole(bytes32 role, address account) internal view {
        if (!hasRole(role, account)) {
            revert(
                string(
                    abi.encodePacked(
                        "AccessControl: account ",
                        Strings.toHexString(uint160(account), 20),
                        " is missing role ",
                        Strings.toHexString(uint256(role), 32)
                    )
                )
            );
        }
    }

    /**
     * @dev Returns the admin role that controls `role`. See {grantRole} and
     * {revokeRole}.
     *
     * To change a role's admin, use {_setRoleAdmin}.
     */
    function getRoleAdmin(bytes32 role) public view override returns (bytes32) {
        return _roles[role].adminRole;
    }

    /**
     * @dev Grants `role` to `account`.
     *
     * If `account` had not been already granted `role`, emits a {RoleGranted}
     * event.
     *
     * Requirements:
     *
     * - the caller must have ``role``'s admin role.
     */
    function grantRole(bytes32 role, address account) public virtual override onlyRole(getRoleAdmin(role)) {
        _grantRole(role, account);
    }

    /**
```

```
 * @dev Revokes `role` from `account`.
 *
 * If `account` had been granted `role`, emits a {RoleRevoked} event.
 *
 * Requirements:
 *
 * - the caller must have ``role``'s admin role.
 */
function revokeRole(bytes32 role, address account) public virtual override onlyRole(getRoleAdmin(role)) {
    _revokeRole(role, account);
}

/**
 * @dev Revokes `role` from the calling account.
 *
 * Roles are often managed via {grantRole} and {revokeRole}: this function's
 * purpose is to provide a mechanism for accounts to lose their privileges
 * if they are compromised (such as when a trusted device is misplaced).
 *
 * If the calling account had been granted `role`, emits a {RoleRevoked}
 * event.
 *
 * Requirements:
 *
 * - the caller must be `account`.
 */
function renounceRole(bytes32 role, address account) public virtual override {
    require(account == _msgSender(), "AccessControl: can only renounce roles for self");

    _revokeRole(role, account);
}

/**
 * @dev Grants `role` to `account`.
 *
 * If `account` had not been already granted `role`, emits a {RoleGranted}
 * event. Note that unlike {grantRole}, this function doesn't perform any
 * checks on the calling account.
 *
 * [WARNING]
 * ====
 * This function should only be called from the constructor when setting
 * up the initial roles for the system.
 *
 * Using this function in any other way is effectively circumventing the admin
```

```
     * system imposed by {AccessControl}.
     * ====
     */
    function _setupRole(bytes32 role, address account) internal virtual {
        _grantRole(role, account);
    }

    /**
     * @dev Sets `adminRole` as ``role``'s admin role.
     *
     * Emits a {RoleAdminChanged} event.
     */
    function _setRoleAdmin(bytes32 role, bytes32 adminRole) internal virtual {
        bytes32 previousAdminRole = getRoleAdmin(role);
        _roles[role].adminRole = adminRole;
        emit RoleAdminChanged(role, previousAdminRole, adminRole);
    }

    function _grantRole(bytes32 role, address account) private {
        if (!hasRole(role, account)) {
            _roles[role].members[account] = true;
            emit RoleGranted(role, account, _msgSender());
        }
    }

    function _revokeRole(bytes32 role, address account) private {
        if (hasRole(role, account)) {
            _roles[role].members[account] = false;
            emit RoleRevoked(role, account, _msgSender());
        }
    }
}
```

ERC721URIStorage.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "../ERC721.sol";

/**
 * @dev ERC721 token with storage based token URI management.
 */
abstract contract ERC721URIStorage is ERC721 {
    using Strings for uint256;

    // Optional mapping for token URIs
    mapping(uint256 => string) private _tokenURIs;

    /**
     * @dev See {IERC721Metadata-tokenURI}.
     */
    function tokenURI(uint256 tokenId) public view virtual override returns (string memory) {
        require(_exists(tokenId), "ERC721URIStorage: URI query for nonexistent token");

        string memory _tokenURI = _tokenURIs[tokenId];
        string memory base = _baseURI();

        // If there is no base URI, return the token URI.
        if (bytes(base).length == 0) {
            return _tokenURI;
        }
        // If both are set, concatenate the baseURI and tokenURI (via abi.encodePacked).
        if (bytes(_tokenURI).length > 0) {
            return string(abi.encodePacked(base, _tokenURI));
        }

        return super.tokenURI(tokenId);
    }

    /**
     * @dev Sets `_tokenURI` as the tokenURI of `tokenId`.
     *
     * Requirements:
     *
     * - `tokenId` must exist.
     */
    function _setTokenURI(uint256 tokenId, string memory _tokenURI) internal virtual {
        require(_exists(tokenId), "ERC721URIStorage: URI set of nonexistent token");
        _tokenURIs[tokenId] = _tokenURI;
    }

    /**
     * @dev Destroys `tokenId`.
     * The approval is cleared when the token is burned.
     *
     * Requirements:
     *
     * - `tokenId` must exist.
     *
     * Emits a {Transfer} event.
     */
    function _burn(uint256 tokenId) internal virtual override {
        super._burn(tokenId);

        if (bytes(_tokenURIs[tokenId]).length != 0) {
            delete _tokenURIs[tokenId];
        }
    }
}
```

ERC721Enumerable.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "../ERC721.sol";
import "./IERC721Enumerable.sol";

/**
 * @dev This implements an optional extension of {ERC721} defined in the EIP that adds
 * enumerability of all the token ids in the contract as well as all token ids owned by each
 * account.
 */
abstract contract ERC721Enumerable is ERC721, IERC721Enumerable {
    // Mapping from owner to list of owned token IDs
    mapping(address => mapping(uint256 => uint256)) private _ownedTokens;

    // Mapping from token ID to index of the owner tokens list
    mapping(uint256 => uint256) private _ownedTokensIndex;

    // Array with all token ids, used for enumeration
    uint256[] private _allTokens;

    // Mapping from token id to position in the allTokens array
    mapping(uint256 => uint256) private _allTokensIndex;

    /**
     * @dev See {IERC165-supportsInterface}.
     */
    function supportsInterface(bytes4 interfaceId) public view virtual override(IERC165, ERC721) returns (bool) {
        return interfaceId == type(IERC721Enumerable).interfaceId || super.supportsInterface(interfaceId);
    }

    /**
     * @dev See {IERC721Enumerable-tokenOfOwnerByIndex}.
     */
    function tokenOfOwnerByIndex(address owner, uint256 index) public view virtual override returns (uint256) {
        require(index < ERC721.balanceOf(owner), "ERC721Enumerable: owner index out of bounds");
        return _ownedTokens[owner][index];
    }

    /**
     * @dev See {IERC721Enumerable-totalSupply}.
     */
```

```
function totalSupply() public view virtual override returns (uint256) {
    return _allTokens.length;
}

/**
 * @dev See {IERC721Enumerable-tokenByIndex}.
 */
function tokenByIndex(uint256 index) public view virtual override returns (uint256) {
    require(index < ERC721Enumerable.totalSupply(), "ERC721Enumerable: global index out of bounds");
    return _allTokens[index];
}

/**
 * @dev Hook that is called before any token transfer. This includes minting
 * and burning.
 *
 * Calling conditions:
 *
 * - When `from` and `to` are both non-zero, ``from``'s `tokenId` will be
 * transferred to `to`.
 * - When `from` is zero, `tokenId` will be minted for `to`.
 * - When `to` is zero, ``from``'s `tokenId` will be burned.
 * - `from` cannot be the zero address.
 * - `to` cannot be the zero address.
 *
 * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
 */
function _beforeTokenTransfer(
    address from,
    address to,
    uint256 tokenId
) internal virtual override {
    super._beforeTokenTransfer(from, to, tokenId);

    if (from == address(0)) {
        _addTokenToAllTokensEnumeration(tokenId);
    } else if (from != to) {
        _removeTokenFromOwnerEnumeration(from, tokenId);
    }
    if (to == address(0)) {
        _removeTokenFromAllTokensEnumeration(tokenId);
    } else if (to != from) {
        _addTokenToOwnerEnumeration(to, tokenId);
    }
}
```

```
/**
 * @dev Private function to add a token to this extension's ownership-tracking data structures.
 * @param to address representing the new owner of the given token ID
 * @param tokenId uint256 ID of the token to be added to the tokens list of the given address
 */
function _addTokenToOwnerEnumeration(address to, uint256 tokenId) private {
    uint256 length = ERC721.balanceOf(to);
    _ownedTokens[to][length] = tokenId;
    _ownedTokensIndex[tokenId] = length;
}

/**
 * @dev Private function to add a token to this extension's token tracking data structures.
 * @param tokenId uint256 ID of the token to be added to the tokens list
 */
function _addTokenToAllTokensEnumeration(uint256 tokenId) private {
    _allTokensIndex[tokenId] = _allTokens.length;
    _allTokens.push(tokenId);
}

/**
 * @dev Private function to remove a token from this extension's ownership-tracking data structures. Note that
 * while the token is not assigned a new owner, the `_ownedTokensIndex` mapping is _not_ updated: this allows for
 * gas optimizations e.g. when performing a transfer operation (avoiding double writes).
 * This has O(1) time complexity, but alters the order of the _ownedTokens array.
 * @param from address representing the previous owner of the given token ID
 * @param tokenId uint256 ID of the token to be removed from the tokens list of the given address
 */
function _removeTokenFromOwnerEnumeration(address from, uint256 tokenId) private {
    // To prevent a gap in from's tokens array, we store the last token in the index of the token to delete, and
    // then delete the last slot (swap and pop).

    uint256 lastTokenIndex = ERC721.balanceOf(from) - 1;
    uint256 tokenIndex = _ownedTokensIndex[tokenId];

    // When the token to delete is the last token, the swap operation is unnecessary
    if (tokenIndex != lastTokenIndex) {
        uint256 lastTokenId = _ownedTokens[from][lastTokenIndex];

        _ownedTokens[from][tokenIndex] = lastTokenId; // Move the last token to the slot of the to-delete token
        _ownedTokensIndex[lastTokenId] = tokenIndex; // Update the moved token's index
    }

    // This also deletes the contents at the last position of the array
```

```
        delete _ownedTokensIndex[tokenId];
        delete _ownedTokens[from][lastTokenIndex];
    }

    /**
     * @dev Private function to remove a token from this extension's token tracking data structures.
     * This has O(1) time complexity, but alters the order of the _allTokens array.
     * @param tokenId uint256 ID of the token to be removed from the tokens list
     */
    function _removeTokenFromAllTokensEnumeration(uint256 tokenId) private {
        // To prevent a gap in the tokens array, we store the last token in the index of the token to delete, and
        // then delete the last slot (swap and pop).

        uint256 lastTokenIndex = _allTokens.length - 1;
        uint256 tokenIndex = _allTokensIndex[tokenId];

        // When the token to delete is the last token, the swap operation is unnecessary. However, since this occurs so
        // rarely (when the last minted token is burnt) that we still do the swap here to avoid the gas cost of adding
        // an 'if' statement (like in _removeTokenFromOwnerEnumeration)
        uint256 lastTokenId = _allTokens[lastTokenIndex];

        _allTokens[tokenIndex] = lastTokenId; // Move the last token to the slot of the to-delete token
        _allTokensIndex[lastTokenId] = tokenIndex; // Update the moved token's index

        // This also deletes the contents at the last position of the array
        delete _allTokensIndex[tokenId];
        _allTokens.pop();
    }
}
```

ERC721.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "./IERC721.sol";
import "./IERC721Receiver.sol";
import "./extensions/IERC721Metadata.sol";
import "../../utils/Address.sol";
import "../../utils/Context.sol";
import "../../utils/Strings.sol";
import "../../utils/introspection/ERC165.sol";

/**
 * @dev Implementation of https://eips.ethereum.org/EIPS/eip-721[ERC721] Non-Fungible Token Standard, including
 * the Metadata extension, but not including the Enumerable extension, which is available separately as
 * {ERC721Enumerable}.
 */
contract ERC721 is Context, ERC165, IERC721, IERC721Metadata {
    using Address for address;
    using Strings for uint256;

    // Token name
    string private _name;

    // Token symbol
    string private _symbol;

    // Mapping from token ID to owner address
    mapping(uint256 => address) private _owners;

    // Mapping owner address to token count
    mapping(address => uint256) private _balances;

    // Mapping from token ID to approved address
    mapping(uint256 => address) private _tokenApprovals;

    // Mapping from owner to operator approvals
    mapping(address => mapping(address => bool)) private _operatorApprovals;

    /**
     * @dev Initializes the contract by setting a `name` and a `symbol` to the token collection.
     */
    constructor(string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
    }

    /**
     * @dev See {IERC165-supportsInterface}.
     */
    function supportsInterface(bytes4 interfaceId) public view virtual override(ERC165, IERC165) returns (bool) {
        return
            interfaceId == type(IERC721).interfaceId ||
            interfaceId == type(IERC721Metadata).interfaceId ||
            super.supportsInterface(interfaceId);
    }

    /**
     * @dev See {IERC721-balanceOf}.
     */
    function balanceOf(address owner) public view virtual override returns (uint256) {
        require(owner != address(0), "ERC721: balance query for the zero address");
        return _balances[owner];
    }

    /**
     * @dev See {IERC721-ownerOf}.
     */
    function ownerOf(uint256 tokenId) public view virtual override returns (address) {
        address owner = _owners[tokenId];
        require(owner != address(0), "ERC721: owner query for nonexistent token");
        return owner;
    }

    /**
     * @dev See {IERC721Metadata-name}.
     */
    function name() public view virtual override returns (string memory) {
        return _name;
    }

    /**
     * @dev See {IERC721Metadata-symbol}.
     */
    function symbol() public view virtual override returns (string memory) {
        return _symbol;
    }
```

```
/**
 * @dev See {IERC721Metadata-tokenURI}.
 */
function tokenURI(uint256 tokenId) public view virtual override returns (string memory) {
    require(_exists(tokenId), "ERC721Metadata: URI query for nonexistent token");

    string memory baseURI = _baseURI();
    return bytes(baseURI).length > 0 ? string(abi.encodePacked(baseURI, tokenId.toString())) : "";
}

/**
 * @dev Base URI for computing {tokenURI}. If set, the resulting URI for each
 * token will be the concatenation of the `baseURI` and the `tokenId`. Empty
 * by default, can be overriden in child contracts.
 */
function _baseURI() internal view virtual returns (string memory) {
    return "";
}

/**
 * @dev See {IERC721-approve}.
 */
function approve(address to, uint256 tokenId) public virtual override {
    address owner = ERC721.ownerOf(tokenId);
    require(to != owner, "ERC721: approval to current owner");

    require(
        _msgSender() == owner || isApprovedForAll(owner, _msgSender()),
        "ERC721: approve caller is not owner nor approved for all"
    );

    _approve(to, tokenId);
}

/**
 * @dev See {IERC721-getApproved}.
 */
function getApproved(uint256 tokenId) public view virtual override returns (address) {
    require(_exists(tokenId), "ERC721: approved query for nonexistent token");

    return _tokenApprovals[tokenId];
}

/**
 * @dev See {IERC721-setApprovalForAll}.
```

```
     */
    function setApprovalForAll(address operator, bool approved) public virtual override {
        require(operator != _msgSender(), "ERC721: approve to caller");

        _operatorApprovals[_msgSender()][operator] = approved;
        emit ApprovalForAll(_msgSender(), operator, approved);
    }

    /**
     * @dev See {IERC721-isApprovedForAll}.
     */
    function isApprovedForAll(address owner, address operator) public view virtual override returns (bool) {
        return _operatorApprovals[owner][operator];
    }

    /**
     * @dev See {IERC721-transferFrom}.
     */
    function transferFrom(
        address from,
        address to,
        uint256 tokenId
    ) public virtual override {
        //solhint-disable-next-line max-line-length
        require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer caller is not owner nor approved");

        _transfer(from, to, tokenId);
    }

    /**
     * @dev See {IERC721-safeTransferFrom}.
     */
    function safeTransferFrom(
        address from,
        address to,
        uint256 tokenId
    ) public virtual override {
        safeTransferFrom(from, to, tokenId, "");
    }

    /**
     * @dev See {IERC721-safeTransferFrom}.
     */
    function safeTransferFrom(
        address from,
        address to,
        uint256 tokenId,
        bytes memory _data
    ) public virtual override {
        require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer caller is not owner nor approved");
        _safeTransfer(from, to, tokenId, _data);
    }

    /**
     * @dev Safely transfers `tokenId` token from `from` to `to`, checking first that contract recipients
     * are aware of the ERC721 protocol to prevent tokens from being forever locked.
     *
     * `_data` is additional data, it has no specified format and it is sent in call to `to`.
     *
     * This internal function is equivalent to {safeTransferFrom}, and can be used to e.g.
     * implement alternative mechanisms to perform token transfer, such as signature-based.
     *
     * Requirements:
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `tokenId` token must exist and be owned by `from`.
     * - If `to` refers to a smart contract, it must implement {IERC721Receiver-onERC721Received}, which is called upon
a safe transfer.
     *
     * Emits a {Transfer} event.
     */
    function _safeTransfer(
        address from,
        address to,
        uint256 tokenId,
        bytes memory _data
    ) internal virtual {
        _transfer(from, to, tokenId);
        require(_checkOnERC721Received(from, to, tokenId, _data), "ERC721: transfer to non ERC721Receiver
implementer");
    }

    /**
     * @dev Returns whether `tokenId` exists.
     *
     * Tokens can be managed by their owner or approved accounts via {approve} or {setApprovalForAll}.
     *
     * Tokens start existing when they are minted (`_mint`),
     * and stop existing when they are burned (`_burn`).
     */
    function _exists(uint256 tokenId) internal view virtual returns (bool) {
        return _owners[tokenId] != address(0);
    }

    /**
     * @dev Returns whether `spender` is allowed to manage `tokenId`.
     *
     * Requirements:
     *
     * - `tokenId` must exist.
     */
    function _isApprovedOrOwner(address spender, uint256 tokenId) internal view virtual returns (bool) {
        require(_exists(tokenId), "ERC721: operator query for nonexistent token");
        address owner = ERC721.ownerOf(tokenId);
        return (spender == owner || getApproved(tokenId) == spender || isApprovedForAll(owner, spender));
    }

    /**
     * @dev Safely mints `tokenId` and transfers it to `to`.
     *
     * Requirements:
     *
     * - `tokenId` must not exist.
     * - If `to` refers to a smart contract, it must implement {IERC721Receiver-onERC721Received}, which is called upon
a safe transfer.
     *
     * Emits a {Transfer} event.
     */
    function _safeMint(address to, uint256 tokenId) internal virtual {
        _safeMint(to, tokenId, "");
    }

    /**
     * @dev Same as {xref-ERC721-_safeMint-address-uint256-}[`_safeMint`], with an additional `data` parameter which is
     * forwarded in {IERC721Receiver-onERC721Received} to contract recipients.
     */
    function _safeMint(
        address to,
        uint256 tokenId,
        bytes memory _data
    ) internal virtual {
        _mint(to, tokenId);
        require(
            _checkOnERC721Received(address(0), to, tokenId, _data),
            "ERC721: transfer to non ERC721Receiver implementer"
        );
    }

    /**
     * @dev Mints `tokenId` and transfers it to `to`.
     *
     * WARNING: Usage of this method is discouraged, use {_safeMint} whenever possible
     *
     * Requirements:
     *
     * - `tokenId` must not exist.
     * - `to` cannot be the zero address.
     *
     * Emits a {Transfer} event.
     */
    function _mint(address to, uint256 tokenId) internal virtual {
        require(to != address(0), "ERC721: mint to the zero address");
        require(!_exists(tokenId), "ERC721: token already minted");

        _beforeTokenTransfer(address(0), to, tokenId);

        _balances[to] += 1;
        _owners[tokenId] = to;

        emit Transfer(address(0), to, tokenId);
    }

    /**
     * @dev Destroys `tokenId`.
     * The approval is cleared when the token is burned.
     *
     * Requirements:
     *
     * - `tokenId` must exist.
     *
     * Emits a {Transfer} event.
     */
    function _burn(uint256 tokenId) internal virtual {
        address owner = ERC721.ownerOf(tokenId);

        _beforeTokenTransfer(owner, address(0), tokenId);

        // Clear approvals
        _approve(address(0), tokenId);
        _balances[owner] -= 1;
        delete _owners[tokenId];

        emit Transfer(owner, address(0), tokenId);
    }

    /**
     * @dev Transfers `tokenId` from `from` to `to`.
     *  As opposed to {transferFrom}, this imposes no restrictions on msg.sender.
     *
     * Requirements:
     *
     * - `to` cannot be the zero address.
     * - `tokenId` token must be owned by `from`.
     *
     * Emits a {Transfer} event.
     */
    function _transfer(
        address from,
        address to,
        uint256 tokenId
    ) internal virtual {
        require(ERC721.ownerOf(tokenId) == from, "ERC721: transfer of token that is not own");
        require(to != address(0), "ERC721: transfer to the zero address");

        _beforeTokenTransfer(from, to, tokenId);

        // Clear approvals from the previous owner
        _approve(address(0), tokenId);

        _balances[from] -= 1;
        _balances[to] += 1;
        _owners[tokenId] = to;

        emit Transfer(from, to, tokenId);
    }

    /**
     * @dev Approve `to` to operate on `tokenId`
     *
     * Emits a {Approval} event.
     */
    function _approve(address to, uint256 tokenId) internal virtual {
        _tokenApprovals[tokenId] = to;
        emit Approval(ERC721.ownerOf(tokenId), to, tokenId);
    }

    /**
     * @dev Internal function to invoke {IERC721Receiver-onERC721Received} on a target address.
     * The call is not executed if the target address is not a contract.
     *
     * @param from address representing the previous owner of the given token ID
     * @param to target address that will receive the tokens
     * @param tokenId uint256 ID of the token to be transferred
     * @param _data bytes optional data to send along with the call
     * @return bool whether the call correctly returned the expected magic value
     */
    function _checkOnERC721Received(
        address from,
        address to,
        uint256 tokenId,
        bytes memory _data
    ) private returns (bool) {
        if (to.isContract()) {
            try IERC721Receiver(to).onERC721Received(_msgSender(), from, tokenId, _data) returns (bytes4 retval) {
                return retval == IERC721Receiver.onERC721Received.selector;
            } catch (bytes memory reason) {
                if (reason.length == 0) {
                    revert("ERC721: transfer to non ERC721Receiver implementer");
                } else {
                    assembly {
                        revert(add(32, reason), mload(reason))
                    }
                }
            }
        } else {
            return true;
        }
    }

    /**
     * @dev Hook that is called before any token transfer. This includes minting
     * and burning.
     *
     * Calling conditions:
     *
     * - When `from` and `to` are both non-zero, ``from``'s `tokenId` will be
     * transferred to `to`.
     * - When `from` is zero, `tokenId` will be minted for `to`.
```

```
     * - When `to` is zero, ``from``'s `tokenId` will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
     */
    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 tokenId
    ) internal virtual {}
}
```

IERC721Enumerable.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "../IERC721.sol";

/**
 * @title ERC-721 Non-Fungible Token Standard, optional enumeration extension
 * @dev See https://eips.ethereum.org/EIPS/eip-721
 */
interface IERC721Enumerable is IERC721 {
    /**
     * @dev Returns the total amount of tokens stored by the contract.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns a token ID owned by `owner` at a given `index` of its token list.
     * Use along with {balanceOf} to enumerate all of ``owner``'s tokens.
     */
    function tokenOfOwnerByIndex(address owner, uint256 index) external view returns (uint256 tokenId);

    /**
     * @dev Returns a token ID at a given `index` of all the tokens stored by the contract.
     * Use along with {totalSupply} to enumerate all tokens.
     */
    function tokenByIndex(uint256 index) external view returns (uint256);
}
```

ERC165.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "./IERC165.sol";

/**
 * @dev Implementation of the {IERC165} interface.
 *
 * Contracts that want to implement ERC165 should inherit from this contract and override {supportsInterface} to check
 * for the additional interface id that will be supported. For example:
 *
 * ```solidity
 * function supportsInterface(bytes4 interfaceId) public view virtual override returns (bool) {
 *     return interfaceId == type(MyInterface).interfaceId || super.supportsInterface(interfaceId);
 * }
 * ```
 *
 * Alternatively, {ERC165Storage} provides an easier to use but more expensive implementation.
 */
abstract contract ERC165 is IERC165 {
    /**
     * @dev See {IERC165-supportsInterface}.
     */
    function supportsInterface(bytes4 interfaceId) public view virtual override returns (bool) {
        return interfaceId == type(IERC165).interfaceId;
    }
}
```

Strings.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

/**
 * @dev String operations.
 */
library Strings {
    bytes16 private constant _HEX_SYMBOLS = "0123456789abcdef";

    /**
     * @dev Converts a `uint256` to its ASCII `string` decimal representation.
     */
    function toString(uint256 value) internal pure returns (string memory) {
        // Inspired by OraclizeAPI's implementation - MIT licence
        // https://github.com/oraclize/ethereum-
api/blob/b42146b063c7d6ee1358846c198246239e9360e8/oraclizeAPI_0.4.25.sol

        if (value == 0) {
            return "0";
        }
        uint256 temp = value;
        uint256 digits;
        while (temp != 0) {
            digits++;
            temp /= 10;
        }
        bytes memory buffer = new bytes(digits);
        while (value != 0) {
            digits -= 1;
            buffer[digits] = bytes1(uint8(48 + uint256(value % 10)));
            value /= 10;
        }
        return string(buffer);
    }

    /**
     * @dev Converts a `uint256` to its ASCII `string` hexadecimal representation.
     */
    function toHexString(uint256 value) internal pure returns (string memory) {
        if (value == 0) {
            return "0x00";
        }
        uint256 temp = value;
        uint256 length = 0;
        while (temp != 0) {
            length++;
            temp >>= 8;
        }
        return toHexString(value, length);
    }

    /**
     * @dev Converts a `uint256` to its ASCII `string` hexadecimal representation with fixed length.
     */
    function toHexString(uint256 value, uint256 length) internal pure returns (string memory) {
        bytes memory buffer = new bytes(2 * length + 2);
        buffer[0] = "0";
        buffer[1] = "x";
        for (uint256 i = 2 * length + 1; i > 1; --i) {
            buffer[i] = _HEX_SYMBOLS[value & 0xf];
            value >>= 4;
        }
        require(value == 0, "Strings: hex length insufficient");
        return string(buffer);
    }
}
```

Context.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

/**
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }
    function _msgData() internal view virtual returns (bytes calldata) {
        return msg.data;
    }
}
```

Address.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies on extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        uint256 size;
        assembly {
            size := extcodesize(account)
        }
        return size > 0;
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-
pattern[checks-effects-interactions pattern].
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain `call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding-
and-decoding-functions[`abi.decode`].
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
        return functionCall(target, data, "Address: low-level call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
     * `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(
        address target,
        bytes memory data,
        uint256 value
    ) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
    }

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`], but
     * with `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(
        address target,
        bytes memory data,
        uint256 value,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");
        require(isContract(target), "Address: call to non-contract");

        (bool success, bytes memory returndata) = target.call{value: value}(data);
        return verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data) internal view returns (bytes memory) {
        return functionStaticCall(target, data, "Address: low-level static call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal view returns (bytes memory) {
        require(isContract(target), "Address: static call to non-contract");

        (bool success, bytes memory returndata) = target.staticcall(data);
        return verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory) {
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(isContract(target), "Address: delegate call to non-contract");

        (bool success, bytes memory returndata) = target.delegatecall(data);
        return verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Tool to verifies that a low level call was successful, and revert if it wasn't, either by bubbling the
     * revert reason using the provided one.
     *
     * _Available since v4.3._
     */
    function verifyCallResult(
        bool success,
        bytes memory returndata,
        string memory errorMessage
    ) internal pure returns (bytes memory) {
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}
```

IERC721Metadata.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "../IERC721.sol";

/**
 * @title ERC-721 Non-Fungible Token Standard, optional metadata extension
 * @dev See https://eips.ethereum.org/EIPS/eip-721
 */
interface IERC721Metadata is IERC721 {
    /**
     * @dev Returns the token collection name.
     */
    function name() external view returns (string memory);

    /**
     * @dev Returns the token collection symbol.
     */
    function symbol() external view returns (string memory);

    /**
     * @dev Returns the Uniform Resource Identifier (URI) for `tokenId` token.
     */
    function tokenURI(uint256 tokenId) external view returns (string memory);
}
```

IERC721Receiver.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

/**
 * @title ERC721 token receiver interface
 * @dev Interface for any contract that wants to support safeTransfers
 * from ERC721 asset contracts.
 */
interface IERC721Receiver {
    /**
     * @dev Whenever an {IERC721} `tokenId` token is transferred to this contract via {IERC721-safeTransferFrom}
     * by `operator` from `from`, this function is called.
     *
     * It must return its Solidity selector to confirm the token transfer.
     * If any other value is returned or the interface is not implemented by the recipient, the transfer will be
reverted.
     *
     * The selector can be obtained in Solidity with `IERC721.onERC721Received.selector`.
     */
    function onERC721Received(
        address operator,
        address from,
        uint256 tokenId,
        bytes calldata data
    ) external returns (bytes4);
}
```

IERC721.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "../../utils/introspection/IERC165.sol";

/**
 * @dev Required interface of an ERC721 compliant contract.
 */
interface IERC721 is IERC165 {
    /**
     * @dev Emitted when `tokenId` token is transferred from `from` to `to`.
     */
    event Transfer(address indexed from, address indexed to, uint256 indexed tokenId);

    /**
     * @dev Emitted when `owner` enables `approved` to manage the `tokenId` token.
     */
    event Approval(address indexed owner, address indexed approved, uint256 indexed tokenId);

    /**
     * @dev Emitted when `owner` enables or disables (`approved`) `operator` to manage all of its assets.
     */
    event ApprovalForAll(address indexed owner, address indexed operator, bool approved);

    /**
     * @dev Returns the number of tokens in ``owner``'s account.
     */
    function balanceOf(address owner) external view returns (uint256 balance);

    /**
     * @dev Returns the owner of the `tokenId` token.
     *
     * Requirements:
     *
     * - `tokenId` must exist.
     */
    function ownerOf(uint256 tokenId) external view returns (address owner);

    /**
     * @dev Safely transfers `tokenId` token from `from` to `to`, checking first that contract recipients
     * are aware of the ERC721 protocol to prevent tokens from being forever locked.
     *
     * Requirements:
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `tokenId` token must exist and be owned by `from`.
     * - If the caller is not `from`, it must be have been allowed to move this token by either {approve} or
{setApprovalForAll}.
     * - If `to` refers to a smart contract, it must implement {IERC721Receiver-onERC721Received}, which is called upon
a safe transfer.
     *
     * Emits a {Transfer} event.
     */
    function safeTransferFrom(
        address from,
        address to,
        uint256 tokenId
    ) external;

    /**
     * @dev Transfers `tokenId` token from `from` to `to`.
     *
     * WARNING: Usage of this method is discouraged, use {safeTransferFrom} whenever possible.
     *
     * Requirements:
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `tokenId` token must be owned by `from`.
     * - If the caller is not `from`, it must be approved to move this token by either {approve} or
{setApprovalForAll}.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(
        address from,
        address to,
        uint256 tokenId
    ) external;

    /**
     * @dev Gives permission to `to` to transfer `tokenId` token to another account.
     * The approval is cleared when the token is transferred.
     *
     * Only a single account can be approved at a time, so approving the zero address clears previous approvals.
     *
```

```
 * Requirements:
 *
 * - The caller must own the token or be an approved operator.
 * - `tokenId` must exist.
 *
 * Emits an {Approval} event.
 */
function approve(address to, uint256 tokenId) external;

/**
 * @dev Returns the account approved for `tokenId` token.
 *
 * Requirements:
 *
 * - `tokenId` must exist.
 */
function getApproved(uint256 tokenId) external view returns (address operator);

/**
 * @dev Approve or remove `operator` as an operator for the caller.
 * Operators can call {transferFrom} or {safeTransferFrom} for any token owned by the caller.
 *
 * Requirements:
 *
 * - The `operator` cannot be the caller.
 *
 * Emits an {ApprovalForAll} event.
 */
function setApprovalForAll(address operator, bool _approved) external;

/**
 * @dev Returns if the `operator` is allowed to manage all of the assets of `owner`.
 *
 * See {setApprovalForAll}
 */
function isApprovedForAll(address owner, address operator) external view returns (bool);

/**
 * @dev Safely transfers `tokenId` token from `from` to `to`.
 *
 * Requirements:
 *
 * - `from` cannot be the zero address.
 * - `to` cannot be the zero address.
 * - `tokenId` token must exist and be owned by `from`.
```

```
     * - If the caller is not `from`, it must be approved to move this token by either {approve} or
{setApprovalForAll}.
     * - If `to` refers to a smart contract, it must implement {IERC721Receiver-onERC721Received}, which is called upon
a safe transfer.
     *
     * Emits a {Transfer} event.
     */
    function safeTransferFrom(
        address from,
        address to,
        uint256 tokenId,
        bytes calldata data
    ) external;
}
```

IAccessControl.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

/**
 * @dev External interface of AccessControl declared to support ERC165 detection.
 */
interface IAccessControl {
    /**
     * @dev Emitted when `newAdminRole` is set as ``role``'s admin role, replacing `previousAdminRole`
     *
     * `DEFAULT_ADMIN_ROLE` is the starting admin for all roles, despite
     * {RoleAdminChanged} not being emitted signaling this.
     *
     * _Available since v3.1._
     */
    event RoleAdminChanged(bytes32 indexed role, bytes32 indexed previousAdminRole, bytes32 indexed newAdminRole);

    /**
     * @dev Emitted when `account` is granted `role`.
     *
     * `sender` is the account that originated the contract call, an admin role
     * bearer except when using {AccessControl-_setupRole}.
     */
    event RoleGranted(bytes32 indexed role, address indexed account, address indexed sender);

    /**
     * @dev Emitted when `account` is revoked `role`.
     *
     * `sender` is the account that originated the contract call:
     *   - if using `revokeRole`, it is the admin role bearer
     *   - if using `renounceRole`, it is the role bearer (i.e. `account`)
     */
    event RoleRevoked(bytes32 indexed role, address indexed account, address indexed sender);

    /**
     * @dev Returns `true` if `account` has been granted `role`.
     */
    function hasRole(bytes32 role, address account) external view returns (bool);

    /**
     * @dev Returns the admin role that controls `role`. See {grantRole} and
     * {revokeRole}.
     *
     * To change a role's admin, use {AccessControl-_setRoleAdmin}.
     */
    function getRoleAdmin(bytes32 role) external view returns (bytes32);

    /**
     * @dev Grants `role` to `account`.
     *
     * If `account` had not been already granted `role`, emits a {RoleGranted}
     * event.
     *
     * Requirements:
     *
     * - the caller must have ``role``'s admin role.
     */
    function grantRole(bytes32 role, address account) external;

    /**
     * @dev Revokes `role` from `account`.
     *
     * If `account` had been granted `role`, emits a {RoleRevoked} event.
     *
     * Requirements:
     *
     * - the caller must have ``role``'s admin role.
     */
    function revokeRole(bytes32 role, address account) external;

    /**
     * @dev Revokes `role` from the calling account.
     *
     * Roles are often managed via {grantRole} and {revokeRole}: this function's
     * purpose is to provide a mechanism for accounts to lose their privileges
     * if they are compromised (such as when a trusted device is misplaced).
     *
     * If the calling account had been granted `role`, emits a {RoleRevoked}
     * event.
     *
     * Requirements:
     *
     * - the caller must be `account`.
     */
    function renounceRole(bytes32 role, address account) external;
}
```

IERC165.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

/**
 * @dev Interface of the ERC165 standard, as defined in the
 * https://eips.ethereum.org/EIPS/eip-165[EIP].
 *
 * Implementers can declare support of contract interfaces, which can then be
 * queried by others ({ERC165Checker}).
 *
 * For an implementation, see {ERC165}.
 */
interface IERC165 {
    /**
     * @dev Returns true if this contract implements the interface defined by
     * `interfaceId`. See the corresponding
     * https://eips.ethereum.org/EIPS/eip-165#how-interfaces-are-identified[EIP section]
     * to learn more about how these ids are created.
     *
     * This function call must use less than 30 000 gas.
     */
    function supportsInterface(bytes4 interfaceId) external view returns (bool);
}
```

Appendix E - Resources Used

Amended Complaint [Dkt 24], dated March 2, 2022

Twitter Feeds from @basicdotspace, @MasonRothschild, @ILYYWNFT

5 brands already boldly embracing the metaverse. (n.d.). The Drum. Retrieved July 24, 2022, from https://www.thedrum.com/news/2022/01/17/5-brands-already-boldly-embracing-the-metaverse

10. Setting fees on secondary sales. (n.d.). OpenSea Developer Documentation. Retrieved July 6, 2022, from https://docs.opensea.io/docs/10-setting-fees-on-secondary-sales

Decentraland. (2018, February 7). *Display an NFT in a scene*. Decentraland. https://docs.decentraland.org/development-guide/display-a-certified-nft/

Dolce&Gabbana enters the Metaverse. (n.d.). Dolce & Gabbana. Retrieved July 25, 2022, from https://world.dolcegabbana.com/discover/dolcegabbana-enters-the-metaverse/

Entriken, W., Shirley, D., Evans, J., & Sachs, N. (2018, January 24). *EIP-721: Non-Fungible Token Standard*. Ethereum Improvement Proposals. https://eips.ethereum.org/EIPS/eip-721

etherscan.io. (n.d.-a). *Ether Daily Price (USD) Chart | Etherscan*. Ethereum (ETH) Blockchain Explorer. Retrieved July 29, 2022, from http://etherscan.io/chart/etherprice

etherscan.io. (n.d.-b). *Ethereum (ETH) Blockchain Explorer*. Ethereum (ETH) Blockchain Explorer. Retrieved July 28, 2022, from http://etherscan.io/

Group, E. V. (2019, July 22). Why do People Spend Billions on Virtual Clothes? *EVG Virtual*. https://medium.com/evg-virtual/why-do-people-spend-billions-on-virtual-clothes-504ec1601521

MetaBirkins by Mason Rothschild. (n.d.). MetaBirkins. Retrieved July 1, 2022, from http://metabirkins.com/

METAVERSE & NFT | GUCCI® VAULT US. (n.d.). Retrieved July 25, 2022, from https://vault.gucci.com/en-US/story/metaverse

Moss, T. (n.d.). *A $300,000 Dolce & Gabbana Tiara You Can Only Wear in the Metaverse*. WSJ. Retrieved July 25, 2022, from https://www.wsj.com/articles/a-300-000-dolce-gabbana-tiara-you-can-only-wear-in-the-metaverse-11648817988

Nast, C. (2022, March 9). *Gucci goes deeper into the metaverse for next NFT project*. Vogue Business. https://www.voguebusiness.com/technology/gucci-goes-deeper-into-the-metaverse-for-next-nft-project

Rarible.com cuts marketplace fees to 1% for each side. (2022, June 20). Rarible Blog. https://rarible.com/blog/lower-fees/

Shop, R. S. (n.d.). *Rolling Stone x Bored Ape Yacht Club Limited-Edition Zine*. Rolling Stone Shop. Retrieved July 20, 2022, from https://shop.rollingstone.com/products/rolling-stone-x-bored-ape-yacht-club-special-collectors-edition-zine

Sotheby's Debuts NFT Exhibition IRL And, Of Course, In The Metaverse. (2021, June 9). Jing Culture and Commerce. https://jingculturecommerce.com/sothebys-natively-digital-nft-exhibition/

Team, E. (n.d.). *Top 10 Fashion Brands Embracing the Metaverse*. ProjectPractical.Com. Retrieved July 24, 2022, from https://www.projectpractical.com/top-10-fashion-brands-embracing-the-metaverse/

Weirdos. (n.d.). Retrieved July 1, 2022, from https://rarible.com/ilyyw/items

What are service and creator fees? (n.d.). OpenSea. Retrieved July 8, 2022, from https://support.opensea.io/hc/en-us/articles/1500011590241-What-are-service-and-creator-fees-

What are the meta-universe's wearables? (n.d.). NonFungible.Com. Retrieved July 22, 2022, from https://nonfungible.com/news/metaverse/what-are-the-meta-universes-wearables

Wallet Addresses involved with the Metabirkins NFT Project:

0xa4464043350b1bd4f11ea345db9e1da2ff7b7dfb

0xa53b1fb8fb30d8c4992d99fa6b8234cd8eb20f23

0xeaaeac965449d2426f6f793770b4f3560eeb7c0f

0x416c9f987be51072e2d5194d360be544d402838b

0xe7591048e2bf6b46f4b195d448f86ac5c4992f01

0x7a71217edb34f53ebb7311d06f9887d4d0214feb

0x4cbcb4efff4c0d2b4eb148d7c7591bf167887831

0x6d76467ca225c30c03dc285bf4c208fdb45d286c

0x7a6397713631c3b51625ddbb11fc96e669f184b3

0xc5f802a45de6af0f3fab6726dc8f466fe089ea16

0x56caf3bc01f69b4ffe3625dbfcb563bde9270f6a

0x72a3162c47d1c7efd3a425410a4931b5dce7d386

0x865a98cb3ca1a22de8a5840dd99a8e98c9e5172d

0x246667ab3dbaa438cf321dd03e347601e74796c0

0xcc6d4546f57ae7d37a2acb17d07228e0d8439e6c

0x5ee409ab7dc23936e817b7153e08d5f512ebd485

0xa061982d4b087d911db8399a641187df945d48d0

0x7bd7d33d6395573324455d81ac0fe3a391e15407

0x8ac02f38443d40ebaee940619b431e7025534184

0xe5859cbc7a5c954d33480e67266c2bbc919a966e

0xde7f82ce6055b79b0d121cbf998f318d21cc75ca

0x4cb4c2f595e737ffa135773cdda8ee482ba52a89

0xc57112fb1872130a85ecf29877dd96042572a027

0x514a8bddbb1afbe1360e49d9702fb3917e66a4d4

0x55d6d25ae0b1280392f98cd30724defb1fea849b

0x424255a97d18926b24b446deaf76cda5e686fce1

0x261e6e7aaacfcde75535fe96943f35bcd86b84c7

0x9199aa7f8715121a7aa8b7d2e87b89f05196b6ac

0xdbcc6698d4686ee3fba49c2245072460594efe6e

0x7e69c45edde9c72a1a121a3ca68218b3c4db5fd6

0xa784779bd895b2db4c0009a7468f090012e12ff9

0xe7fd709e6f774ff277e984ff0ab64630cf34a783

0x23274a30baa2d8b3b15464857acbe744d1a2b592

0xd8443486d005558ab49bc288d425612dbed6a958

0xa244f5434a77e09f282e727aa1317517b0c944f1

0x0d205aa4f1512b08b953a79f1b98be5992dc9565

0xd9dd93fb4b1708b44b27bf73fc84f0602080a903

0x86b669137b45246b386d7517ae6defad84e23e64

0xd22a2f097c35dfe83285ca4c121f54809be89dd3

0xb455f2f2fe79543261a59dc2475ace3f5dbaf86b

0xd6f0e7d50b6f39796c38681ade7bf5a93be501b0

0xa71caa1f6791300a91b09fafcf5b410f20576fcb

0x18b4dbc8ec72f95b83b3e80e8932b2453b578cab

0xbbb677a94eda9660832e9944353dd6e814a45705

0xaa15a96fba6c84737e632a93140d9f549f55338f

0x9c097a3124495f129265c8a1f44c61be55422139

0x14413004541be2f1de60355a2fd2f62c38fd6d92

0x607783b409b4db847d36ae9f9dce910d35597a73

0xc477e5b2cb6d972dad2b2fe385c9dae42333d193

0x47b2b5dc779b6109ddc397b510dc72d894d116f3

0x9ea29e7b0e5367709ffee125681a6be65ff07b21

0x1dba27c0eac19c0ccfc5703af60d61d9f8e36981

0x38b9c791a6aa299eed24e60feadbf438198ee26c

0xaa37eb2841f4d1c4b283cc21800e2616d669b0b9

0xd8891897f4b87af7a79b116ed170d976c7646fcc

0x5e2edc210e572812976151c2da9d2c1c5f10ef85

0x79e087a1957fccf93649d3732f9193f0fc8c5138

0xec8d6078d4566d190fde8a707bc2a27071280002

0xa77be93ea4c6fd083d0e967951bee7de725ada8a

0xe5df0517420999c690459aed4ce897232a65cc7d

0x24a1c1260cdcb7aa99807151ea70e235329693ea

0x140a3394819762e4d485f1f1f8fc9ade92014800

0x3b0f3cfbbbd9402cb95391fe6a1bc64267db4f07

0x443dce4789f715257f71aecf866dffe6611458c3

0x802ce3834cd488f31d58163eb24ddd7ed2cb836a

0x61453a32c3e8cfdb70f5ec5cb9bd654a1fa46321

0x269dded1c8e9391073ff7853cbd8ba5d6a043d93

0x3d6b0258391b4f7e1769e29a255b4f4f712aeab8

0x1723a88158118ad1ae03a873b2df1e8de4fb921e

0x3008e2cc65c47f0c30e0555e099d80f1dd9a2a63

0x09e5775097abb28205726af83e2bc606dd6cc5f5

0x734b955dbdec0fea40511556fa07f6b6f5a1ee7d

0x59fcf421e1b1efdf8189a91c220b245be0dec9de

0x5c6893cbfdc696dcc91cbd78f65c72efff91d554

0x278db415b3c969e789e1ec9e80e1e001a5dcee82

0x13fc16d47ce2578b5ab6a0eafc6854f62851086b

0xc2db6199e024f368c1827c07fe356e8e5e4ece00

0x7b724878f967f3731a0ac6a52745a05269b0e9db

0x7c45387374deb534a417e3a7399b04f69e43e0b2

0xe468ce99444174bd3bbbed09209577d25d1ad673

0xaa8e798172b5ebc0ad29099ec971fb78e6df0166

0x8090dd2831092d07c013e8252cd7a19f9149e2ea

0xa6c80dc48783eb9d3e3ca4ea1e1dad61e5adeb2a

```
0x93d95608bc5fc0331ef12bd50d6e0a333ca3fb07

0x5edb7dc2e3fc7f4f42c0e4283dcb067252d89291

0xa5cc132432c1f7d59aa10c0af7ef994c86907f56

0x018e196d257c02ea961fb29c78477967227586f7

0xfc000a4dab3bcc69e58e4332953d2d58848f5f81

0x9d2b10f35d57c522ccb644518a61c646caa9fcf6

0xf31f8d16601c7cc8796499354e96de0a1200ba76

0xed19dc5a34e3523d9cdfad74517e7725baea15e7

0x7061a2ba0e52e0050bd03b3d84b66d6079b631c7

0x7822db7499bc453acf26bed1c3101fd311cee08e

0x998f33ec8ce47981318d5d8ef38e783e2279ff78

0xa75a18d018be21c6d19ca7e3df0d2f7d73f0bd88

0x305fa474af0433b1e68a768a32898ab34edc2cec

0xb16746458242cfefeb06ab986dce789dc21f1ab4

0x1fb8409569e1239f68003c1d665b9c7014e2514a

0xcd7f43bebae54f92a236a079a212e6ec7eaa2332

0x2c398a281a22b74e65652c9be97c2e393401acf9

0x03e12307bdbe8c9fce363161b58c1ab3072fa145

0xfd44be061efb17c000d7233f0dec1a0c814b994e

0xf1b8231826cc4bfa8664d5aa7b507567178f9cfa

0x862ffe14939c39c6d7ac18ceb4728ebf8257ef90
```

0x525f901de6f9c09b0bbd037d6134e9f009170850

0xa9edcc8d5a0d0f22eef74cbd21d70baf12180317

0x0c9b5e263ba6e8c3474f6449ab73082b01d3a0e8

0xdb06ec407313c0035db0d574c707b2c6a450c7db

0x87728f756760daeb53cb9e2c913c6e67efea4979

0x457d8ad97f4a5ca23c3b8a56044a666b78587bbb

0x540e9966f424b1122eb79945bf876226963edd1d

0x87d580a0b983b26cd4ce3c191e807f596f3793bf

0xae71d942f256d0e1480c75be3386298cde20b00b

0x7e5313af2e16e6e0855865a16bb08fbe86157dd1

0x6a71f74809c4759f5d0396625bd9bc9a409e32af

0xbc07118e364374b4476f2a3219df8b558f05fe7f

0x2c960bb932c2f1e2f868592d348c108272e7dd87

0x9b3596cb07db659db65f2f6dceffb9af376fc1c1

0x04881cad0eca331568ab0cb43c69a509bc92556e

0xa532669f826363d9a3957979b0c40d567d7cc1c6

0x8bd88a6f95a4dab3ac4d8fdadee296bfff4a0333

0x2cad3fe928e0e338d5ea0a79646d61b43ff38ab9

0xa9d1f18c98e00a00a6a97f96d2ba78ed96480203

0x6144b6360afb2b713bc65e6660d0f75943e99e10

0x78e37504ac03ce61ac5d5978592c10b38a8dbcff

0x67f3a567762f212822ccb47c348c3ce3d9f6db7a

0x56f8503ebdee6214f5616bb79537fff806cc4c52

0x243dc5d02592a989fe1b809d263f432e6632b999

0xa4a68fe2abacda9a0a08cff0bd2185009020db49

0xb5c105ed531bc6ad1c2c38372dc797d47c01bb67

0xccf1d3cced6fb98972b38c17941ff6cbd5a0f86d

0x22032ba84376443d54a59f20346c6475acad0945

0x787a296f1b7646453cba1df483801e3dbba29396

0x8c35933c469406c8899882f5c2119649cd5b617f

0xde5fb7260d57ba0e4f7f1beab6217f5a6cedfe85

0x82ed2ab8e5b709db88f963c490a37b11df7bc526

0x3d32673ae56045dbd2704e7692d2b50b59f14534

0x48889a2cea22b794d7bcbfd156a404ba37b1776c

0x2004c725dbfe6d7a4cd0c37e976c335b825a02ae

0x0cc1a222be2128686ee8a5837cd82811f58ca16e

0x775936c4dca762d38e329930c60dc4a71b724ca1

0x6d33ad572e072b97ffea834cb7be668490afe6cd

0x1616b4c7cdb4093befbcca62f3198993327a8e9e

0x12f37431468eb75c2a825e2cf8fde773ad94c8ea

0x4fcea0a8fc0df0ce0d653ece0c79e2b75a30f4d0

0xc8490e455a02669a8ce2609e04f4c2f82f2c7227

```
0x21556681c4762aac131cc2684bf838d794e374be

0xcda04b6bfb3ffff7413d2d8a613d3f23dd10cb1f

0xc1dcad513b8e066a3f27dd5cc3edf0a920645bc4

0x694de865c83e90ebd8876f98b6a9fe5dc2182169

0x1350a306b27b1f650423757e0ca50d65aa4f2093

0xf63a4d80d28f8c304892b989c6ad315cb4951d5f

0x39cbb7e024baf3301951c01664cd73b39629fb81

0x58c17f3ded16162f559bc590b7cff18a657ddaf4

0x9f43ef42b1960548bd9c986a90392e8fee0693b1

0x901b8708f9516c84d777a74f142e44b89f437b93

0xafd1a6e363c2cbcb72dbb583aa3084a10a13b800

0xe5fb12703a7c83859d84cba18958484c5a1309b8

0x3f8b5e08d8d0eedb36e13c3ed6bbc7350acceaa0

0x7d0c87c3574cb72c96a128666ffa27cdecc8aaef

0x4455a6e247eaa29208a697809176b1c01a1ab958

0xf3fd33ce765115a2d1e70acb10e8299b9b92b8bb

0x9b2175beab7265491d71f1fe00fb2d5a949f9ba8

0xf0d6999725115e3ead3d927eb3329d63afaec09b

0x0674751942f1a35597c15f608615aa1ae0a10169

0xaf56c40b392f9ec9679fde332a7a81de11ac6efd

0xc63a38574693285e1a9acfaa2334122a7c733cfa
```

0xd0d2eac7a9961aed27b4c41420b779b93229e427

0xcd9d4dd3e066a77039607a82588365b75a1f4778

0x34e8c8c4dce046a30e0844f492826c61578a3248

0x85d0894d28c2bfc4ef332fa7548b0b9417d470e4